SRI SIVASUBRAMANIYA NADAR COLLEGE OF ENGINEERING

(AN AUTONOMOUS INSTITUTION,
AFFILIATED TO ANNA UNIVERSITY)

Rajiv Gandhi Salai (OMR), Kalavakkam - 603 110.

LABORATORY RECORD

NAME : Krithika Swaminathan

Reg. No. : 205001057

Dept. : CSE Sem. : IV Sec. : A

SSN

Assignment 1 – Study of System Calls and System Commands

Date: 07/03/2022

1. Observation – Study of System calls and System commands

SYSTEM COMMANDS:

cp: Command used to copy files or directories from a source to a destination

Options:

-i : Used to prompt before overwrite (overrides a previous -n option).

-r : Recursive - option used to copy directories recursively.

-s : Symbolic link - option used to make symbolic links.

Syntax: cp <option> <source(s)> <destination>

Example: cp -r dir1 dir2

cp assgn.txt dir2

mv: Command used to move (or) rename files

Options:

-i : Interactive - option used to prompt before overwriting a file.

-b : Backup - option used to make a backup of each existing destination file, but doesn't accept arguments.

-f : Used to remove prompting before overwriting.

Syntax: mv <option> <source> <destination>

mv <oldname> <newname>

Example: mv ex1.txt assgn.txt

mv assgn.txt ../Desktop

ls: Command used to list information about the contents of a directory

Options:

-l : Long list order: used to display directory contents in long list order.

-R : Recursive - used to display the subdirectories recursively.

-r : Reverse - used to display the contents of the directory in reverse order.

Syntax: ls <option> <directory>

Example: ls -R Desktop

ls -l Downloads/dir1

grep: Command used to print the lines matching a given pattern from a file
If no files are specified, grep searches the standard input.
Options:
-v : Invert match - used to print non-matching lines.
-n : Line number - used to print the matching lines with each line prefixed by its line number from the file.
-c : Count - used to suppress the normal output and print the count of matching lines from the input file.
Syntax: grep <option> 'pattern' <filename>
Example: grep -n "System" assgn.txt

chmod: Command used to change the permissions for a file or directory by changing the file mode bits
Options:
-f :Suppress the error messages.
-R : used to change files and directories recursively.
-v : output a diagnostic for every file processed.
Syntax: chmod <option> MODE <file/directory>
Example: chmod u=rwx assgn.txt //using symbolic code
chmod 755 assgn.txt //using octal code

cat: Command used to concatenate files and print on standard output
When no file is given, it reads standard output.
Options:
-n : used to number all the output lines.
-T : used to display tab characters as '^I'.
-s : used to suppress repeated empty output lines.
Syntax: cat <option> <file>
Example: cat assgn.txt

mkdir: Command used to create a new directory
Options:
-m : used to set the file mode of the new directory.
-p : used to make parent directories as needed.
-v : used to print a message for each created directory.

Syntax: mkdir <option> <directoryname>
Example: mkdir newdir

rm: Command used to remove files or directories
By default, it does not remove directories.
Options:
-i : used to prompt before removing a file or directory.
-d : used to remove empty directories.
-r : used to remove directories and their contents recursively.
Syntax: rm <option> <file/directory>
Example: rm -d newdir

rmdir: Command used to remove empty directories
Options:
-p : Parents - used to remove a given directory including its parent directories.
-v : output a diagnostic for every directory processed.
Syntax: rmdir <option> <directory>
Example: rmdir newdir

wc: Command used to print the count of the number of lines, words and characters in a file
Options:
-c : used to print the number of characters in a file.
-w : used to print the number of words in a file.
-l : used to print the number of lines in the file.
Syntax: wc <option> <file>
Example: wc -w assgn.txt

who: Command used to show which user is currently logged into the system
Options:
-q : used to print all login names and number of users logged in.
-m : used to print only hostname and the user associated with stdin.
-r : used to print current run level.
Syntax: who <option> <file/arguments>
Example: who -q

head: Command used to output the first n lines of a file to the standard output
When no file is given, it reads the standard output.
Options:
-n : prints the first n lines from the top of a file.
-v : used to always print headers giving filenames.
-c : prints the first n bytes from the beginning of the file.
Syntax: head <option> <filename>
Example: head -6 assgn.txt

tail: Command used to output the last n lines of a file to standard output
When no file is given, it reads standard output.
Options:
-n : prints the last n lines from the file.
-c : prints the last n bytes from the file.
-v : used to always output headers giving filenames.
Syntax: tail <option> <filename>
Example: tail -4 assgn.txt

nl: Command used to write each file to standard output, with line numbers added
When no file is given, it reads from standard input.
Options:
-p : do not reset line numbers at logical pages.
-n: used to insert line numbers according to a given format.
-v : first line number on each logical page.
Syntax: nl <option> <file>
Example: nl -p assgn.txt

awk: A pattern scanning and text processing language consisting of a sequence of pattern-action pairs acting on a file
Options:
-F : used to set the field separator to a given value.
-f : used to read the program text from a file instead of the command line.
-v : used to assign a value to a program variable.
Syntax: awk <options> ‘Program text’ <filename>

Example: awk '{print $0}' assgn.txt

SYSTEM CALLS:

fork():

This function is used to create a new process by duplicating the existing process from which it is called. The existing process from which this function is called is called parent process and the newly created process is called child process.

Header file: unistd.h

Syntax: pid_t fork(void);

Arguments: None

Return type: On success, PID is returned in the parent and 0 is returned in the child. On failure, -1 is returned to the parent.

execl():

This function is used to execute a file which is residing in an active process. It replaces the previous executable file and the new file is executed.

Header file: unistd.h

Syntax: int execl(const char* path, const char* arg……./* (char*) NULL */);

Arguments: Path of the executable binary file and arguments (i.e. -lh/home) to be passed to the executable file followed by NULL.

Return type: Returns -1 if error occurs(failure), else does not return anything(success).

getpid():

This function returns the process id of the calling process. It is often used by routines that generate unique temporary filenames.

Header file: unistd.h

Syntax: pid_t getpid(void);

Arguments: None

Return type: Always returns the process ID of the calling process. It never throws any errors, so it is always successful.

getppid():

This function is used to return the process id of the parent of the calling process.

Header file: unistd.h

Syntax: pid_t getppid(void);
Arguments: None
Return type: Always returns the process ID of the parent of the calling process. It never throws any errors, so it is always successful.

exit():
This function is used to terminate the normal process and return the value of status to the parent.
Header file: stdlib.h
Syntax: void exit(int status);
Arguments: An integer value of the exit status.
Return type: The exit() function does not return anything, either during success or failure.

wait():
This function suspends the execution of the calling process until one of its child processes terminates.
Header file: sys/wait.h
Syntax: pid_t wait(int *status);
Arguments: Address of the status(integer value).
Return type: On success, it returns the process id of the terminated child. On failure, it returns -1.

close():
This function is used to close a file descriptor, so that it longer refers to any file and can be reused. Any record locks held on the file it was associated with, and owned by the process, are removed.
Header file: unistd.h
Syntax: int close(int fd);
Arguments: Value of file descriptor ‘fd’.
Return type: On success, it returns 0. On failure, it returns -1.

opendir():
This function opens a directory stream corresponding to the directory name, and returns a pointer to the directory stream. The stream is positioned at the first entry in the directory.

Header file: dirent.h
Syntax: DIR *opendir(const char *name);
Arguments: Name of the directory
Return type: On success, it returns a pointer to the directory stream. On failure, it returns NULL.

readdir():
This function returns a pointer to the dirent structure representing the next directory entry in the directory stream pointed to by dirp.
Header file: dirent.h
Syntax: struct dirent *readdir(DIR *dirp);
Arguments: Directory pointer ‘dirp’
Return type: On success, it returns a pointer to a dirent structure. On failure, it returns NULL.

open():
This function returns a file descriptor for the given file pathname, for use in subsequent system calls. The file descriptor returned by a successful call will be the lowest numbered file descriptor not currently open for the process.
Header file: fcntl.h
Syntax: int open(const char *pathname, int flags);
Return type: On success, it returns the file descriptor(integer). On failure, it returns -1.

read():
This function is used to read upto ‘count’ bytes from a given file descriptor ‘fd’ into the buffer starting at ‘buf’.
Header file: unistd.h
Syntax: ssize_t read(int fd, void *buf, size_t count);
Arguments: A file descriptor, buffer and number of bytes to be read.
Return type: On success, it returns the number of bytes read. On failure, it returns -1.

write():
This function is used to write upto ‘count’ bytes from the buffer starting at ‘buf’ to the file referred to by the file descriptor ‘fd’.
Header file: unistd.h
Syntax: ssize_t write(int fd, const void *buf, size_t count);

Arguments: File descriptor, buffer and number of bytes to be written.
Return type: On success, it returns the number of bytes written. On failure, it returns -1.

creat():
This function is equivalent to the open() system call. It is actually a redundant function which has been primarily included for historical purposes since many applications depend on it. It creates a new file or rewrites an existing one.
Header file: fcntl.h
Syntax: int creat(const char *pathname, mode_t mode);
Arguments: Pathname of the file, mode
Return type: On success, it returns the file descriptor. On failure, it returns -1.

sleep():
This function is used to delay or pause for a specified amount of time (in seconds).
Header file: unistd.h
Syntax: unsigned int sleep(unsigned int seconds);
Arguments: Delay time in seconds.
Return type: Zero if the requested time has elapsed, or the number of seconds left to sleep, if the call was interrupted by a signal handler

2. Develop a C program to understand the working of fork().

Aim:

To develop a C program to understand the working of fork().

Algorithm:

1. Start
2. Get a process id by forking.
3. If the process id is 0, then proceed to the child process. This will print a chosen unique message a certain number of times.
4. If not, then proceed to the parent process. This will print a different identifying message a given number of times.
5. Stop

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include  <sys/types.h>

#define   MAX_COUNT  200


void  ChildProcess(void);
void  ParentProcess(void);

void  main(void) {
        pid_t  pid;
        pid = fork();
        if (pid == 0)
                ChildProcess();
        else
                ParentProcess();
        }

void  ChildProcess(void) {
        int   i;
        for (i = 1; i <= MAX_COUNT; i++)
                printf("\tThis line is from child, value = %d\n", i);
        printf("\t*** Child process is done ***\n");
        }

void  ParentProcess(void) {
```

```
int   i;
for (i = 1; i <= MAX_COUNT; i++)
        printf("This line is from parent, value = %d\n", i);
printf("*** Parent is done ***\n");
}
```

Output:

```
kri@kri-ubuntu:~/workspace$ gcc -o fork fork_call.c
kri@kri-ubuntu:~/workspace$ ./fork
This line is from parent, value = 1
This line is from parent, value = 2
This line is from parent, value = 3
This line is from parent, value = 4
This line is from parent, value = 5
        This line is from child, value = 1
        This line is from child, value = 2
        This line is from child, value = 3
        This line is from child, value = 4
This line is from parent, value = 6
        This line is from child, value = 5
This line is from parent, value = 7
        This line is from child, value = 6
This line is from parent, value = 8
        This line is from child, value = 7
This line is from parent, value = 9
        This line is from child, value = 8
This line is from parent, value = 10
        This line is from child, value = 9
This line is from parent, value = 11
        This line is from child, value = 10
This line is from parent, value = 12
        This line is from child, value = 11
This line is from parent, value = 13
        This line is from child, value = 12
This line is from parent, value = 14
        This line is from child, value = 13
This line is from parent, value = 15
        This line is from child, value = 14
This line is from parent, value = 16
        This line is from child, value = 15
This line is from parent, value = 17
        This line is from child, value = 16
This line is from parent, value = 18
        This line is from child, value = 17
This line is from parent, value = 19
        This line is from child, value = 18
This line is from parent, value = 20
        This line is from child, value = 19
This line is from parent, value = 21
        This line is from child, value = 20
This line is from parent, value = 22
        This line is from child, value = 21
This line is from parent, value = 23
        This line is from child, value = 22
This line is from parent, value = 24
        This line is from child, value = 23
This line is from parent, value = 25
        This line is from child, value = 24
This line is from parent, value = 26
        This line is from child, value = 25
This line is from parent, value = 27
        This line is from child, value = 26
This line is from parent, value = 28
        This line is from child, value = 27
This line is from parent, value = 29
        This line is from child, value = 28
This line is from parent, value = 30
        This line is from child, value = 29
This line is from parent, value = 31
        This line is from child, value = 30
This line is from parent, value = 32
        This line is from child, value = 31
This line is from parent, value = 33
        This line is from child, value = 32
This line is from parent, value = 34
        This line is from child, value = 33
This line is from parent, value = 35
        This line is from child, value = 34
This line is from parent, value = 36
        This line is from child, value = 35
This line is from parent, value = 37
        This line is from child, value = 36
This line is from parent, value = 38
```

```
        This line is from child, value = 37
This line is from parent, value = 39
        This line is from child, value = 38
This line is from parent, value = 40
        This line is from child, value = 39
This line is from parent, value = 41
        This line is from child, value = 40
This line is from parent, value = 42
        This line is from child, value = 41
This line is from parent, value = 43
        This line is from child, value = 42
This line is from parent, value = 44
        This line is from child, value = 43
This line is from parent, value = 45
        This line is from child, value = 44
This line is from parent, value = 46
        This line is from child, value = 45
This line is from parent, value = 47
        This line is from child, value = 46
This line is from parent, value = 48
        This line is from child, value = 47
This line is from parent, value = 49
        This line is from child, value = 48
This line is from parent, value = 50
```

.

.

.

```
        This line is from child, value = 183
This line is from parent, value = 184
        This line is from child, value = 184
This line is from parent, value = 185
        This line is from child, value = 185
This line is from parent, value = 186
        This line is from child, value = 186
This line is from parent, value = 187
        This line is from child, value = 187
This line is from parent, value = 188
        This line is from child, value = 188
This line is from parent, value = 189
        This line is from child, value = 189
This line is from parent, value = 190
        This line is from child, value = 190
This line is from parent, value = 191
        This line is from child, value = 191
This line is from parent, value = 192
        This line is from child, value = 192
This line is from parent, value = 193
        This line is from child, value = 193
This line is from parent, value = 194
        This line is from child, value = 194
This line is from parent, value = 195
        This line is from child, value = 195
This line is from parent, value = 196
        This line is from child, value = 196
This line is from parent, value = 197
        This line is from child, value = 197
This line is from parent, value = 198
        This line is from child, value = 198
This line is from parent, value = 199
        This line is from child, value = 199
This line is from parent, value = 200
        This line is from child, value = 200
*** Parent is done ***
        *** Child process is done ***
```

3. Develop a C program using system calls to open a file, read the contents of the same, display it and close the file. Use command line arguments to pass the file name to the program.

Aim:

To develop a C program using system calls to open a file, read the contents of the same, display it and close the file.

Algorithm:

1. Start
2. Check if the number of command line arguments is 2. If it is greater than 2, print that there are too many arguments. If it is lesser, then print that an argument is expected.
3. Find the file descriptor for the source file given as the argument. Include an error message for the case when the source file is not found.
4. Save the contents of the source file in a string.
5. Display the contents of the file by printing the string.
6. Close the source file.
7. Stop

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <dirent.h>
#include <fcntl.h>
#include <string.h>

#define MAX_LENGTH 100
#define MAX_BYTES 500

int main (int argc, char* argv[]){
        if (argc == 2){
                char src[MAX_LENGTH];
                strcpy(src,argv[1]);
                int src_fd, src_sz;
                char* content = (char *) calloc(100, sizeof(char));
                printf("\n");

                //find file desciptor for src file
                src_fd = open(src,O_RDONLY);
                //include error msg for case when src file not found
```

```
            if (src_fd < 0) { perror("Command unsuccessful. Source file not found.");
exit(1); }

            //save the contents of src file in a string
            src_sz = read(src_fd,content,MAX_BYTES);
            content[src_sz] = '\0';

            //display the contents of the file by printing the string
            printf("%s\n",content);

            //close src file
            close(src_fd);
            }

        else if (argc > 2){
            printf("Too many arguments supplied.\n");
            }

        else {
            printf("An argument is expected.\n");
            }
        }
```

Output:

```
kri@kri-ubuntu:~/workspace$ gcc -o readf readf.c
kri@kri-ubuntu:~/workspace$ ./readf
An argument is expected.
kri@kri-ubuntu:~/workspace$ ./readf arg1 arg2
Too many arguments supplied.
kri@kri-ubuntu:~/workspace$ ./readf randfile

Command unsuccessful. Source file not found.: No such file or directory
kri@kri-ubuntu:~/workspace$ ./readf foo.txt

This is the first line.
This is the second line.
.
.
.
This is the last line.

kri@kri-ubuntu:~/workspace$
```

Learning outcomes:

- Various system calls and commands were studied.
- The common options used with each command were observed.
- The syntax and header files required for each command were understood.
- The purpose and applications of each command were studied.

Assignment 2 – Simulation of System Commands using System Calls

Date: 14/03/2022

Program 1

Aim:

To develop a C program to simulate the cp command with the -i option using system calls.

Algorithm:

my_cp (-i):

1. Start
2. Check if an option (-i) is included in the command.
3. If the difference between the number of arguments and the number of options is less than three, then print that two arguments are expected. If it is more than three, communicate to the user that there are too many arguments. If it is exactly three, then proceed.
4. If an option is included, then set the 2nd argument as the source file and the 3rd argument as the destination file.
5. If an option is not included, then set the 1st argument as the source file and the 2nd argument as the destination file.
6. Open the source file and locate its file descriptor. If it already exists, proceed. If not, then report that the source file was not found.
7. Copy the contents of the source file to a string.
8. Open the destination file and do one of the following:
    1. If it does not exist, then create a new file and write the contents of the string to it.
    2. If it already exists and there is no (-i) option, then overwrite the contents of the string to the file.
    3. If it already exists and there is a (-i) option, then prompt the user to confirm before overwriting the file.
        If the user says “yes”, then overwrite the file.
        If the user says “no” (or does not say yes), then exit the program without making any changes.
9. Stop

Code:

```
/* C program to simulate the cp [-i] command */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <dirent.h>
#include <fcntl.h>
#include <string.h>

#define MAX_LENGTH 100
#define MAX_BYTES 500

int main (int argc, char* argv[]){
	if (argc == 1) {
		perror("Two arguments expected"); exit(1);
		}

	int opCount = 0;
	if (strcmp(argv[1],"-i") == 0) {
			opCount++;
			}

	if (argc - opCount == 3){
		char src[MAX_LENGTH], dest[MAX_LENGTH], option[2];
		int src_fd, src_sz, dest_fd, dest_sz;
		char *c = (char *) calloc(100, sizeof(char));

		//check if command is to operating in interactive mode
		if (strcmp(argv[1],"-i") == 0) {
			strcpy(src,argv[2]);
			strcpy(dest,argv[3]);

			//check if dest file exists
			dest_fd = open(dest,O_RDONLY);
			//ask the user if dest file is to be overwritten or not
			if (dest_fd >= 0) {
				char ch;
				printf("cp: overwrite '%s'? ",dest);
				scanf("%c",&ch);
				if (ch != 'y' && ch != 'Y')
					exit(1);
				}
			}
		else {
			strcpy(src,argv[1]);
			strcpy(dest,argv[2]);
```

```
                }

                //find file descriptor for src file
                src_fd = open(src,O_RDONLY);
                //include error msg for case when src file not found
                if (src_fd < 0) { perror("Command unsuccessful. Source file not found.");
exit(1); }

                //save the contents of src file in a string
                src_sz = read(src_fd,c,MAX_BYTES);
                c[src_sz] = '\0';

                //close src file
                close(src_fd);

                //create dest file
                dest_fd = open(dest,O_WRONLY | O_CREAT | O_TRUNC,0754);
                //include error msg for case when dest file not created
                if (dest_fd < 0) { perror("Command unsuccessful. Destination file not
created."); exit(1); }

                //write the saved contents of the src file from the string to the dest file
                dest_sz = write(dest_fd,c,strlen(c));

                //close dest file
                close(dest_fd);
                }

        else if (argc - opCount > 3){
                perror("Too many arguments supplied"); exit(1);
                }

        else {
                perror("Two arguments expected"); exit(1);
                }
        }
```

Output:

```
root@hadoop-slave-3:~/krith# gcc -o mycp mycp.c
root@hadoop-slave-3:~/krith# ./mycp
Two arguments expected: Success
root@hadoop-slave-3:~/krith# ./mycp -i
Two arguments expected: Success
root@hadoop-slave-3:~/krith# ./mycp file
Two arguments expected: Success
root@hadoop-slave-3:~/krith# cat> foo.txt
This is the first line.
This is the second line.
This is the third line.
.
.
.
This is the last line.
^C
root@hadoop-slave-3:~/krith# cat file1
cat: file1: No such file or directory
root@hadoop-slave-3:~/krith# ./mycp foo.txt file1
root@hadoop-slave-3:~/krith# cat file1
This is the first line.
This is the second line.
This is the third line.
.
.
.
This is the last line.
root@hadoop-slave-3:~/krith# touch file2
root@hadoop-slave-3:~/krith# ./mycp -i file2 file1
cp: overwrite 'file1'? n
root@hadoop-slave-3:~/krith# cat file1
This is the first line.
This is the second line.
This is the third line.
.
.
.
This is the last line.
root@hadoop-slave-3:~/krith# ./mycp -i file2 file1
cp: overwrite 'file1'? y
root@hadoop-slave-3:~/krith# cat file1
root@hadoop-slave-3:~/krith# ./mycp foo.txt file1
root@hadoop-slave-3:~/krith# cat file1
This is the first line.
This is the second line.
This is the third line.
.
.
.
This is the last line.
root@hadoop-slave-3:~/krith#
```

Program 2

Aim:

To develop a C program to simulate the ls command with the -l option using system calls.

Algorithm:

my_ls (-l):

1. Start
2. Check if an option (-l) is included in the command.
3. Find the difference between the number of arguments and the number of options.
    1. If it is 0, set the current directory to be the directory under consideration.
    2. If it is 1 or more, set each of the directories entered as an argument on the command line to be the set of directories under consideration.
4. For each directory under consideration, get the entries in the directory structure.
5. If option (-l) is included, then list the directory entries along with their file properties in long format. If not, then simply list the directory entries.
6. Stop

Code:

```
/* C program to simulate the ls [-l] command */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <dirent.h>
#include <fcntl.h>
#include <string.h>
#include <time.h>

#define MAX_LENGTH 100
#define MAX_BYTES 500

void ls_dir (char*);
void ls_long (char*);
void ls_file (char*);
void printFileProperties(struct stat);

int main (int argc, char* argv[]) {
```

```
	if (argc == 1) {
		ls_dir(".");
		exit(1);
		}

	int opCount = 0;
	if (strcmp(argv[1],"-l")==0) {
		opCount = 1;
		if (argc-opCount == 1) {
			ls_long(".");
			}

		else if (argc-opCount == 2) {
			ls_long(argv[1+opCount]);
			}

		else {
			for (int i=1+opCount; i<argc; i++) {
				if (i==1+opCount)
					printf("%s: \n",argv[i]);
				else
					printf("\n%s: \n",argv[i]);
				ls_long(argv[i]);
				}
			}
		}

	else {
		if (argc-opCount == 1) {
			//call function to list contents of curr dir
			ls_dir(".");
			}

		else if (argc-opCount == 2) {
			ls_dir(argv[1]);
			}

		else {
			for (int i=1; i<argc; i++) {
				printf("%s: \n",argv[i]);
				ls_dir(argv[i]);
				}
			}
		}
	}

//function to list contents of a given dir
void ls_dir (char* dir) {
	DIR *folder;
```

```
        struct dirent *entry;

        //open given dir
        folder = opendir(dir);
        if (folder == NULL) {
                perror("Could not open directory.");
                exit(1);
                }
        //read and print contents of dir
        while (entry = readdir(folder)) {
                if (strcmp(entry->d_name,".")!=0 && strcmp(entry->d_name,"..")!=0)
                        printf(" %s\n", entry->d_name);
                }

        //close dir
        closedir(folder);
        //printf("\n");
        }

//function to list contents of a given dir in long format
void ls_long (char* dir) {
        DIR *folder;
        struct dirent *entry;
        struct stat fileStats;

        //open given dir
        folder = opendir(dir);
        if (folder == NULL) {
                perror("Could not open directory");
                exit(1);
                }
        //read and print contents of dir
        while (entry = readdir(folder)) {
                if (strcmp(entry->d_name,".")!=0 && strcmp(entry->d_name,"..")!=0) {
                        //ls_file(entry->d_name);
                        stat(entry->d_name,&fileStats);
                        printFileProperties(fileStats);
                        printf("%s\n",entry->d_name);
                        }
                }

        //close dir
        closedir(folder);
        }

//function to print the file properties
void printFileProperties(struct stat stats) {
        struct tm dt;
```

```
        //printf("%ld ", stats.st_ino);

        //printf("%o ", stats.st_mode);

        printf( (S_ISDIR(stats.st_mode)) ? "d" : "-");
        printf( (stats.st_mode & S_IRUSR) ? "r" : "-");
        printf( (stats.st_mode & S_IWUSR) ? "w" : "-");
        printf( (stats.st_mode & S_IXUSR) ? "x" : "-");
        printf( (stats.st_mode & S_IRGRP) ? "r" : "-");
        printf( (stats.st_mode & S_IWGRP) ? "w" : "-");
        printf( (stats.st_mode & S_IXGRP) ? "x" : "-");
        printf( (stats.st_mode & S_IROTH) ? "r" : "-");
        printf( (stats.st_mode & S_IWOTH) ? "w" : "-");
        printf( (stats.st_mode & S_IXOTH) ? "x" : "-");
        printf(" ");

        printf("%ld ", stats.st_nlink);

        printf("%ld ", stats.st_size);

        //get file creation time in seconds and convert seconds to date and time format
        dt = *(gmtime(&stats.st_ctime));
        printf("%d-%d-%d %d:%d:%d ", dt.tm_mday, dt.tm_mon, dt.tm_year + 1900,
dt.tm_hour, dt.tm_min, dt.tm_sec);

        // File modification time
        dt = *(gmtime(&stats.st_mtime));
        printf("%d-%d-%d %d:%d:%d ", dt.tm_mday, dt.tm_mon, dt.tm_year + 1900,
dt.tm_hour, dt.tm_min, dt.tm_sec);
        }
```

Output:

```
root@hadoop-slave-3:~/krith# gcc -o myls myls.c
root@hadoop-slave-3:~/krith# ./myls
 mycp.c
 myls
 foo.txt
 folder
 file1
 mycp
 file2
 myls.c
 mygrep.c
root@hadoop-slave-3:~/krith# ./myls -l
-rw-r--r-- 1 1980 14-2-2022 7:15:35 14-2-2022 7:15:35 mycp.c
-rwxr-xr-x 1 13448 14-2-2022 7:32:41 14-2-2022 7:32:41 myls
-rw-r--r-- 1 102 14-2-2022 7:21:45 14-2-2022 7:21:45 foo.txt
drwxr-xr-x 2 4096 14-2-2022 7:32:18 14-2-2022 7:32:18 folder
-rwxr-xr-- 1 102 14-2-2022 7:23:5 14-2-2022 7:23:5 file1
-rwxr-xr-x 1 13328 14-2-2022 7:20:39 14-2-2022 7:20:39 mycp
-rw-r--r-- 1 0 14-2-2022 7:22:20 14-2-2022 7:22:20 file2
-rw-r--r-- 1 3258 14-2-2022 7:27:56 14-2-2022 7:27:56 myls.c
-rw-r--r-- 1 4251 14-2-2022 7:16:54 14-2-2022 7:16:54 mygrep.
root@hadoop-slave-3:~/krith# ./myls nofolder
Could not open directory.: No such file or directory
root@hadoop-slave-3:~/krith# ./myls -l nofolder
Could not open directory: No such file or directory
root@hadoop-slave-3:~/krith# ./myls folder
 file9
 file8
 file6
 file7
root@hadoop-slave-3:~/krith# ./myls -l folder
---------- 0 0 18-1-1970 14:10:8 18-1-1970 14:10:58 file9
---------- 0 0 18-1-1970 14:10:8 18-1-1970 14:10:58 file8
---------- 0 0 18-1-1970 14:10:8 18-1-1970 14:10:58 file6
---------- 0 0 18-1-1970 14:10:8 18-1-1970 14:10:58 file7
root@hadoop-slave-3:~/krith#
```

Program 3

Aim:

To develop a C program to simulate the grep command with the -c, -n and -v options using system calls.

Algorithm:

my_grep (-c,-n,-v):

1. Start
2. Check if there is at least one argument. If not, exit.
3. Check for options that are included in the command and validate them.
4. If the difference between the number of arguments and the number of options is less than three, report that the number of arguments is less. If not, proceed.
5. Set the argument immediately succeeding the command name (and options, if any) to be the pattern that is to be searched for.
6. Set the rest of the arguments to be the files that are to be searched.
7. Locate the file descriptor of each file. If a file is not found, report an error.
8. Read the contents of each file in a string.
9. Search the string for substrings that match the given pattern.
10. Do one of the following:
    1. If no option is present, display the lines containing matches of the pattern.
    2. If option (-c) is given, display the frequency of pattern matches in the string.
    3. If option (-n) is given, display the lines containing matches of the pattern along with their line numbers.
    4. If option (-v) is given, display the lines which do not contain matches of the pattern.
    5. If a combination of these options is given, display the corresponding combined output as required.
11. Stop

Code:

```
/* C program to simulate the grep [-cvn] command */
#include <stdio.h>
```

```
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <dirent.h>
#include <fcntl.h>
#include <string.h>

#define MAX_LENGTH 100
#define MAX_BYTES 10000
#define MAX_OPTIONS 3

int substring_count (char*, char*);

int main (int argc, char* argv[]){
	if (argc == 1) {
		perror("Two arguments expected"); exit(1);
		}

	int opCount = 0;
	for (int i=1; i<argc && (argv[i][0]=='-' && argv[i][1]!='\0'); i++) {
		opCount++;
		}

	if (argc - opCount >= 3){
		int src_fd, src_sz, num = MAX_OPTIONS;
		char src[MAX_LENGTH], pattern[MAX_LENGTH], options[num];
		char *content = (char *) calloc(MAX_BYTES, sizeof(char));
		memset(options,'-',num*sizeof(options[0]));

		//determine which options have been included
		int k=0;
		for (int i=1; i<argc && (argv[i][0]=='-' && argv[i][1]!='\0'); i++) {
			for (int j=1; argv[i][j]!='\0'; j++) {
				switch (argv[i][j]) {
					case 'n': {
						options[0] = argv[i][j];
						k++; break;
						}
					case 'v': {
						options[1] = argv[i][j];
						k++; break;
						}
					case 'c': {
						options[2] = argv[i][j];
						k++; break;
						}
					default: {
						perror("Invalid option. Exiting..."); exit(1);
```

```
                                        break;
                                        }
                                }
                        }
                }

        //identify the pattern from CLA
        strcpy(pattern,argv[1+opCount]);

        for (int i = opCount+2; i<argc; i++) {
                //identify the filename from CLA
                strcpy(src,argv[i]);

                //find file descriptor for src file
                src_fd = open(src,O_RDONLY);
                //include error msg for case when src file not found
                if (src_fd < 0) { perror("Command unsuccessful"); exit(1); }

                //save the contents of src file in a string
                src_sz = read(src_fd,content,MAX_BYTES);
                content[src_sz] = '\0';

                //close src file
                close(src_fd);

                //display required output
                if (k==0) {
                        //split the content into lines
                        char* line = strtok(content,"\n");
                        while (line != NULL) {
                                //check if pattern is a substring
                                        //note: match regex too
                                        //regcomp and regex functions

                                if (strstr(line,pattern))
                                        printf("%s\n", line);
                                //go to the next line
                                line = strtok(NULL,"\n");
                                }
                        }

                else if (options[2] == 'c') {
                        int count = 0;
                        //split the content into lines
                        char* line = strtok(content,"\n");
                        while (line != NULL) {
                                //check if pattern is a substring
                                if (strstr(line,pattern)) {
                                        count += substring_count(line,pattern);
```

```
                }
            //go to the next line
            line = strtok(NULL,"\n");
            }
        //print the number of occurrences
        printf("%d\n",count);
        }

else if (options[0] == 'n' && options[1] == '-') {
        int n = 0;
        //split the content into tokens on newline
        char* line = strtok(content,"\n");
        while (line != NULL) {
            //count the line numbers
            n++;
            //check if pattern is a substring
            if (strstr(line,pattern))
                printf("%d:%s\n", n,line);
            //go to the next line
            line = strtok(NULL,"\n");
            }
        }

else if (options[0] == 'n' && options[1] == 'v') {
        int n = 0;
        //split the content into tokens on newline
        char* line = strtok(content,"\n");
        while (line != NULL) {
            //count the line numbers
            n++;
            //check if pattern is a substring
            if (!strstr(line,pattern))
                printf("%d:%s\n", n,line);
            //go to the next line
            line = strtok(NULL,"\n");
            }
        }

else if (options[0] == '-' && options[1] == 'v') {
        //split the content into tokens on newline
        char* line = strtok(content,"\n");
        while (line != NULL) {
            //check if pattern is a substring
            if (!strstr(line,pattern))
                printf("%s\n",line);
            //go to the next line
            line = strtok(NULL,"\n");
            }
        }
```

```
                        else { perror("Search unsuccessful"); exit(1); }

                        }
                }

        else {
                perror("At least two arguments expected"); exit(1);
                }
        }

//function to count the frequency of a given substring in a string
int substring_count(char* string, char* substring) {
        int i, j, l1, l2;
        int count = 0;
        int found = 0;

        l1 = strlen(string);
        l2 = strlen(substring);

        for(i = 0; i < l1-l2 + 1; i++) {
                found = 1;
                for(j = 0; j < l2; j++) {
                        if(string[i+j] != substring[j]) {
                                found = 0;
                                break;
                                }
                        }
                if(found) {
                        count++;
                        i = i + l2 -1;
                        }
                }
        return count;
        }
```

Output:

```
root@hadoop-slave-3:~/krith# gcc -o mygrep mygrep.c
root@hadoop-slave-3:~/krith# ./mygrep
Two arguments expected: Success
root@hadoop-slave-3:~/krith# ./mygrep -c
At least two arguments expected: Success
root@hadoop-slave-3:~/krith# ./mygrep line foo.txt
This is the first line.
This is the second line.
This is the third line.
This is the last line.
root@hadoop-slave-3:~/krith# cat foo.txt
This is the first line.
This is the second line.
This is the third line.
.
.
.
This is the last line.
root@hadoop-slave-3:~/krith# ./mygrep -c line foo.txt
4
root@hadoop-slave-3:~/krith# ./mygrep -n line foo.txt
1:This is the first line.
2:This is the second line.
3:This is the third line.
7:This is the last line.
root@hadoop-slave-3:~/krith# ./mygrep -v line foo.txt
.
.
.
root@hadoop-slave-3:~/krith# ./mygrep -nv line foo.txt
4:.
5:.
6:.
root@hadoop-slave-3:~/krith# ./mygrep -n -v line foo.txt
4:.
5:.
6:.
root@hadoop-slave-3:~/krith#
```

Learning outcomes:

- Linux commands were replicated by programming in C.
- The inner workings of the given commands were understood while implementing them.
- Validation of input and error handling was implemented.

Assignment 3 – Implementation of CPU Scheduling Policies: FCFS and SJF (Non-preemptive and Preemptive)

Date: 21/03/2022

Aim:

To develop a menu-driven C program to implement the CPU Scheduling Algorithms FCFS, SJF and SRTF.

Algorithm:

1. Start
2. Declare a structure with elements such as the process name, arrival time, burst time, waiting time and turnaround time.
3. Create a menu with options for the following:
    a) FCFS
    b) SJF
    c) SRTF

    1. FCFS:
        1. Get the details of the processes as input from the user. This includes the arrival time and the burst time of each process.
        2. As the first process does not need to wait, assign the waiting time for the first process as zero.
        3. For every subsequent, set the current waiting time as the sum of the burst time and the waiting time of the previous process.
        4. For the wording target, set the turnaround time as the sum of the waiting time and the burst time.
        5. Compute the average of the total waiting time and the turnaround time for all the processes.

    2. SJF:
        1. Get the details of the processes as input from the user. This includes the arrival time and the burst time of each process.
        2. Sort the processes in the increasing order of their burst times.

3. As the first process does not need to wait, assign the waiting time for the first process as zero.
4. For every subsequent project, set the current waiting time as the sum of the burst time and the waiting time of the previous process.
5. For every process, set the turnaround time as the sum of the waiting time and the burst time.
6. Compute the average of the total waiting time and the turnaround time for all the processes.

3. SRTF:
    1. Repeat the following steps until all the processes have been completed:
        1. Find the process with the minimum remaining time at every single time lap.
        2. Reduce its time by 1.
        3. Check if its remaining time becomes 0.
        4. Increment the counter for process completion.
        5. Calculate the waiting time and turnaround time for each completed process.
        6. Increment time lapsed by 1.
    2. Compute the average of the total waiting time and the turnaround time.

4. Stop

Code:

```
/*C Program to understand and implement CPU Scheduling processes*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_CAP 10
#define MAX_LEN 3
#define LIM 999

struct Schedule {
        char pID[3];
        int atime, btime, ttime, wtime;
        };

void inputProcesses (struct Schedule[],int);
void burstSort (struct Schedule[],int);
```

```
void preemptiveSort (struct Schedule[],int);
void printArray (int[],int);
void calcTimes (struct Schedule[],int);
void printTable (struct Schedule[],int);
void printGantt (struct Schedule[],int);
void printLine();
void printDashLine();
void printShortLine();

int main() {
	printf("\n\t\tCPU SCHEDULING ALGORITHMS\n\n");

	printf("_____MENU_____\n\n");
	printf("1. FCFS\n2. SJF\n3. SRTF\n\n");

	int choice = 1;
	while (choice != 0) {
		printf("\nEnter choice: ");
		scanf("%d",&choice);

		switch(choice) {
			case 1: {
				printf("\n__FCFS__\n");
				struct Schedule fcfs[MAX_CAP];

				int numP;
				printf("Enter no. of processes: ");
				scanf("%d",&numP);

				//get process details
				inputProcesses(fcfs,numP);
				//calculate waiting time and turnabout time
				calcTimes(fcfs,numP);
				//print results
				printTable(fcfs,numP);
				//print gantt chart
				printGantt(fcfs,numP);

				break;
				}

			case 2: {
				printf("\n__SJF__\n");
				struct Schedule sjf[MAX_CAP];

				int numP;
				printf("Enter no. of processes: ");
				scanf("%d",&numP);
```

```
                        //get process details
                        inputProcesses(sjf,numP);
                        //sort according to burst time
                        burstSort(sjf,numP);
                        //calculate wtime and ttime
                        calcTimes(sjf,numP);
                        //print results
                        printTable(sjf,numP);
                        //print gantt chart
                        printGantt(sjf,numP);

                        break;
                        }

                case 3: {
                        printf("\n__SRTF__\n");
                        struct Schedule srtf[MAX_CAP];

                        int numP;
                        printf("Enter no. of processes: ");
                        scanf("%d",&numP);

                        //get process details
                        inputProcesses(srtf,numP);
                        //sort according to shortest remaining time
                        preemptiveSort(srtf,numP);
                        //print results
                        printTable(srtf,numP);

                        break;
                        }

                case 0: {
                        printf("Exiting menu...\n\n");
                        exit(0); break;
                        }

                default: printf("Invalid choice!\n"); break;
                }
        }

    return 0;
    }

void inputProcesses (struct Schedule sc[], int numP) {
    for (int i=0; i<numP; i++) {
        printf("\nEnter processID: ");
        scanf("%s",sc[i].pID);
```

```
                printf("Enter arrival time: ");
                scanf("%d",&sc[i].atime);

                printf("Enter burst time: ");
                scanf("%d",&sc[i].btime);
                }
        }

void calcTimes (struct Schedule sc[], int numP) {
        int wait = 0;
        for (int i=0; i<numP; i++) {
                sc[i].wtime = wait - sc[i].atime;
                if (sc[i].wtime<0) {
                        sc[i].wtime = 0;
                        wait = 0;
                        }
                wait += sc[i].btime;
                sc[i].ttime = sc[i].btime + sc[i].wtime;
                }
        }

void printTable (struct Schedule sc[], int numP) {
        printf("\n__No. of processes: %d__\n",numP);
        float wsum = 0, tsum = 0;
        printLine();
        printf("pID\tA_time\tB_time\tW_time\tT_time\n");
        printDashLine();
        for (int i=0; i<numP; i++) {

printf("%s\t%d\t%d\t%d\t%d\n",sc[i].pID,sc[i].atime,sc[i].btime,sc[i].wtime,sc[i].ttime);

                wsum += sc[i].wtime;
                tsum += sc[i].ttime;
                }
        printLine();
        printf("\tAvg. waiting time: %.2f\n",wsum/numP);
        printf("\tAvg. turnaround time: %.2f\n",tsum/numP);
        printLine();
        printf("\n");
        //printGantt(sc,numP);
        }

void printGantt (struct Schedule sc[], int numP) {
        printf("\nGantt chart:\n");
        int time = 0;
        printShortLine();
        printf("|%d|",time);
        for (int i=0; i<numP; i++) {
                time += sc[i].btime;
```

```
                printf("%s|%d|",sc[i].pID,time);
                }
        printf("\n");
        printShortLine();
        printf("\n");
        }

void burstSort (struct Schedule sc[], int n) {
        for (int i=0; i<n-1; i++) {
                int min_idx = i;
                for (int j=i+1; j<n; j++)
                        if (sc[j].btime < sc[min_idx].btime)
                                min_idx = j;

        struct Schedule temp = sc[min_idx];
                sc[min_idx] = sc[i];
                sc[i] = temp;
                }
        }

void preemptiveSort (struct Schedule sc[], int numP) {
        sc[MAX_CAP-1].btime = LIM;
        int i, smallest, count = 0, newcount = 0, time, context = 0, burst[MAX_CAP];
        //struct Schedule newsc[MAX_CAP];

        for (i=0; i<numP; i++)
                burst[i] = sc[i].btime;

        for (time=0; count!=numP; time++) {
                smallest = MAX_CAP-1;
                for (i=0; i<numP; i++) {
                        if (sc[i].atime<time && sc[i].btime<sc[smallest].btime &&
sc[i].btime>0) {
                                smallest = i;
                                /*printf("%d,%d,%d,%s\n",time,i,smallest,sc[smallest].pID);
                                newsc[newcount] = sc[smallest];
                                newsc[newcount].btime = time - context;
                                newcount++;
                                context = time;*/
                                }
                        }
                sc[smallest].btime--;

                if (sc[smallest].btime == 0) {
                        count++;
                        sc[smallest].wtime = time - sc[smallest].atime - burst[smallest];
                        sc[smallest].ttime = time - sc[smallest].atime;
                        }
                }
```

```
	for (i=0; i<numP; i++)
		sc[i].btime = burst[i];

	//printGantt(newsc,newcount);
	}

void printArray (int arr[], int size) {
	for (int i=0; i < size; i++)
		printf("%d ",arr[i]);
	printf("\n");
	}

void printLine() {
	printf("_____\n");
	}

void printDashLine() {
	printf("----------\n");
	}

void printShortLine() {
	printf("----------\n");
	}
```

Output:

```
kri@kri-ubuntu:~/workspace$ gcc -o cpu cpumenu.c
kri@kri-ubuntu:~/workspace$ ./cpu

                CPU SCHEDULING ALGORITHMS

_____MENU_____

1. FCFS
2. SJF
3. SRTF


Enter choice: 1

__FCFS__
Enter no. of processes: 3

Enter processID: p1
Enter arrival time: 0
Enter burst time: 24

Enter processID: p2
Enter arrival time: 0
Enter burst time: 3

Enter processID: p3
Enter arrival time: 0
Enter burst time: 3
```

```
__No. of processes: 3__
_____________________________________________
pID     A_time  B_time  W_time  T_time
---------------------------------------------
p1      0       24      0       24
p2      0       3       24      27
p3      0       3       27      30
_____________________________________________
        Avg. waiting time: 17.00
        Avg. turnaround time: 27.00
_____________________________________________


Gantt chart:
------------------------
|0|p1|24|p2|27|p3|30|
------------------------
```

```
Enter choice: 1

__FCFS__
Enter no. of processes: 0

__No. of processes: 0__
_____________________________________________
pID     A_time  B_time  W_time  T_time
---------------------------------------------
_____________________________________________
        Avg. waiting time: -nan
        Avg. turnaround time: -nan
_____________________________________________


Gantt chart:
------------------------
|0|
------------------------


Enter choice: 1

__FCFS__
Enter no. of processes: 3

Enter processID: p1
Enter arrival time: 0
Enter burst time: 24

Enter processID: p2
Enter arrival time: 1
Enter burst time: 3

Enter processID: p3
Enter arrival time: 2
Enter burst time: 3
```

```
__No. of processes: 3__
_________________________________________
pID     A_time  B_time  W_time  T_time
-----------------------------------------
p1      0       24      0       24
p2      1       3       23      26
p3      2       3       25      28
_________________________________________
        Avg. waiting time: 16.00
        Avg. turnaround time: 26.00
_________________________________________


Gantt chart:
------------------------
|0|p1|24|p2|27|p3|30|
------------------------
```

```
Enter choice: 2

__SJF__
Enter no. of processes: 4

Enter processID: p1
Enter arrival time: 0
Enter burst time: 6

Enter processID: p2
Enter arrival time: 0
Enter burst time: 8

Enter processID: p3
Enter arrival time: 0
Enter burst time: 7

Enter processID: p4
Enter arrival time: 0
Enter burst time: 3

__No. of processes: 4__
_________________________________________
pID     A_time  B_time  W_time  T_time
-----------------------------------------
p4      0       3       0       3
p1      0       6       3       9
p3      0       7       9       16
p2      0       8       16      24
_________________________________________
        Avg. waiting time: 7.00
        Avg. turnaround time: 13.00
_________________________________________


Gantt chart:
------------------------
|0|p4|3|p1|9|p3|16|p2|24|
------------------------
```

```
Enter choice: 2

__SJF__
Enter no. of processes: 4

Enter processID: p1
Enter arrival time: 0
Enter burst time: 1

Enter processID: p2
Enter arrival time: 1
Enter burst time: 5

Enter processID: p3
Enter arrival time: 2
Enter burst time: 9

Enter processID: p4
Enter arrival time: 3
Enter burst time: 6

__No. of processes: 4__
_________________________________________
pID     A_time  B_time  W_time  T_time
-----------------------------------------
p1      0       1       0       1
p2      1       5       0       5
p4      3       6       3       9
p3      2       9       10      19
_________________________________________
        Avg. waiting time: 3.25
        Avg. turnaround time: 8.50
_________________________________________


Gantt chart:
------------------------
|0|p1|1|p2|6|p4|12|p3|21|
------------------------
```

```
Enter choice: 3

__SRTF__
Enter no. of processes: 4

Enter processID: p1
Enter arrival time: 0
Enter burst time: 8

Enter processID: p2
Enter arrival time: 1
Enter burst time: 4

Enter processID: p3
Enter arrival time: 2
Enter burst time: 9

Enter processID: p4
Enter arrival time: 3
Enter burst time: 5
```

```
__No. of processes: 4__
_____________________________________________
pID     A_time  B_time  W_time  T_time
---------------------------------------------
p1      0       8       9       17
p2      1       4       0       4
p3      2       9       15      24
p4      3       5       2       7
_____________________________________________
        Avg. waiting time: 6.50
        Avg. turnaround time: 13.00
_____________________________________________
```

Learning outcomes:

- The various CPU Scheduling algorithms were understood.
- The CPU Scheduling algorithms, i.e., FCFS, SJF and SRTF were implemented in C.
- Non-preemptive scheduling was understood.

Assignment 4 – Implementation of CPU Scheduling Policies: Priority and Round Robin

Date: 04/04/2022

Aim:

To develop a menu driven C program to implement the CPU Scheduling Algorithms Priority (Non-Preemptive and Preemptive) and Round Robin.

Algorithm:

1. Start
2. Declare a storage structure
3. Create a menu with options for the following
    1. Round Robin

        Get the arrival time and the burst time of each process as input from the user.

        Set the waiting time for all the processes to be 0.

        Repeat the following steps until all the processes have been used. Start with the first process on the list.

        Reduce its remaining type by 1.

        Check if its remaining time becomes 0. If yes, increment the counter for process completion and move on to the next process.

        Check if the allotted quantum for the process has expired. If yes, save the remaining time for the process and move on to the next one.

        If the last process has exhausted its time, lol.

    2. Priority

        Get the arrival time, the burst time and priority of each process as input from the user.

        Set the waiting time for all the processes to be 0.

        Repeat the following steps until all the processes have been used. Start with the first process on the list.

            Find the process with the maximum priority.

            Reduce its remaining type by 1.

            Check if its remaining time becomes 0. If yes, increment the counter for process completion and move on to the next process.

Check if the allotted quantum for the process has expired. If yes, save the remaining time for the process and move on to the next one.

Increase time lapsed by 1.

Calculate the turnaround time for each process, which is the sum of the respective burst and waiting times.

Compute the average waiting time and the average turnaround time.

4. Stop

Code:

```
//Program to understand CPU Scheduling processes
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_CAP 10
#define MAX_LEN 3
#define LIM 999

struct Schedule {
        char pID[3];
        int atime, btime, ttime, wtime;
        int priority;
        };

void inputProcesses (struct Schedule[],int);
void prioritySort (struct Schedule[],int);
void roundrobin (struct Schedule[],int,int);
void printTablePrty (struct Schedule[],int);
void printGantt (struct Schedule[],int);
void printLine();
void printDashLine();
void printShortLine(int);

int main() {
        printf("\n\t\tCPU SCHEDULING ALGORITHMS\n\n");

        printf("_____MENU_____\n\n");
        printf("1. ROUND ROBIN\n2. PRIORITY\n\n");

        int choice = 1;
        while (choice != 0) {
                printf("\nEnter choice: ");
                scanf("%d",&choice);

                switch(choice) {
```

```
                case 1: {
                        printf("\n__ROUND ROBIN__\n");

                        char pID[10][3], porder[MAX_CAP][3]; int ptime[MAX_CAP],
pcount = 0;
                        int i, NOP, sum=0,count=0, y, quant, wt=0, tat=0, rst=0, at[10],
bt[10], temp[10], rt[10], flag[10];
                        float avg_wt, avg_tat, avg_rt;

                        printf("Enter no. of processes: ");
                        scanf("%d",&NOP);

                        printf("Enter time quantum: ");
                        scanf("%d",&quant);

                        //get process details
                        for (i=0; i<NOP; i++) {
                                printf("\nEnter processID: ");
                                scanf("%s",pID[i]);

                                printf("Enter arrival time: ");
                                scanf("%d",&at[i]);

                                printf("Enter burst time: ");
                                scanf("%d",&bt[i]);

                                temp[i] = bt[i];
                                flag[i] = 0;
                                }

                        y = NOP;

                        printf("\n__No. of processes: %d__\n",NOP);
                        printLine();
                        printf("|
pID\t|A_time\t|B_time\t|W_time\t|T_time\t|R_time\t|\n");
                        printDashLine();
                        for (sum=0, i=0; y!=0; ) {
                                if (temp[i] <= quant && temp[i] > 0) {
                                        if (flag[i]==0) {
                                                rt[i] = sum;
                                                flag[i] = 1;
                                                }
                                        sum = sum + temp[i];
                                        strcpy(porder[pcount],pID[i]);
                                        ptime[pcount] = sum;
                                        pcount++;
                                        temp[i] = 0;
                                        count=1;
```

```
                }
            else if(temp[i] > 0) {
                if (flag[i]==0) {
                    rt[i] = sum;
                    flag[i] = 1;
                    }
                temp[i] = temp[i] - quant;
                sum = sum + quant;
                strcpy(porder[pcount],pID[i]);
                ptime[pcount] = sum;
                pcount++;
                }


            if(temp[i]==0 && count==1) {
                y--;
                printf("| %s\t| %d\t| %d\t| %d\t| %d\t|
%d\t|\n",pID[i],at[i],bt[i],sum-at[i]-bt[i],sum-at[i],rt[i]);
                wt = wt+sum-at[i]-bt[i];
                tat = tat+sum-at[i];
                rst = rst+rt[i];
                count =0;
                }

            if(i==NOP-1)
                i=0;
            else if(at[i+1]<=sum)
                i++;
            else
                i=0;
        }

        avg_wt = wt * 1.0/NOP;
        avg_tat = tat * 1.0/NOP;
        avg_rt = rst * 1.0/NOP;

printf("|_____________________________________________________|\n");
        printf("| \tAvg. waiting time: %.2f\t\t\t|\n",avg_wt);
        printf("| \tAvg. turnaround time: %.2f\t\t|\n",avg_tat);
        printf("| \tAvg. response time: %.2f\t\t|\n",avg_rt);

printf("|_____________________________________________________|\n");
        printf("\n");

        printf("\nGantt chart:\n");
        printShortLine(pcount);
        printf("| |");
        for (int i=0; i<pcount; i++) {
            printf("  %s  | |",porder[i]);
```

```
                        }
                    printf("\n");
                    printShortLine(pcount);
                    printf(" 0 ");
                    for (int i=0; i<pcount; i++) {
                        printf("      %d",ptime[i]);
                        }
                    printf("\n\n");

                    break;
                    }

                case 2: {
                    printf("\n__PRIORITY__\n");
                    struct Schedule pr[MAX_CAP];

                    int numP;
                    printf("Enter no. of processes: ");
                    scanf("%d",&numP);

                    //get process details
                    inputProcesses(pr,numP);
                    //sort according to priority
                    prioritySort(pr,numP);
                    //print results
                    printTablePrty(pr,numP);

                    break;
                    }

                case 0: {
                    printf("Exiting menu...\n\n");
                    exit(0); break;
                    }

                default: printf("Invalid choice!\n"); break;
                }
            }

    return 0;
    }

void inputProcesses (struct Schedule sc[], int numP) {
    for (int i=0; i<numP; i++) {
        printf("\nEnter processID: ");
        scanf("%s",sc[i].pID);

        printf("Enter arrival time: ");
        scanf("%d",&sc[i].atime);
```

```
                printf("Enter burst time: ");
                scanf("%d",&sc[i].btime);

                printf("Enter priority: ");
                scanf("%d",&sc[i].priority);
                }
        }

void printTablePrty (struct Schedule sc[], int numP) {
        printf("\n__No. of processes: %d__\n",numP);
        float wsum = 0, tsum = 0;
        printLine();
        printf("| pID\t|A_time\t|B_time\t|Prty\t|W_time\t|T_time\t|\n");
        printDashLine();
        for (int i=0; i<numP; i++) {
                printf("| %s\t| %d\t| %d\t| %d\t| %d\t|
%d\t|\n",sc[i].pID,sc[i].atime,sc[i].btime,sc[i].priority,sc[i].wtime,sc[i].ttime);

                wsum += sc[i].wtime;
                tsum += sc[i].ttime;
                }
        printf("|_____________________________________________________|\n");
        printf("| \tAvg. waiting time: %.2f\t\t\t|\n",wsum/numP);
        printf("| \tAvg. turnaround time: %.2f\t\t|\n",tsum/numP);
        printf("|_____________________________________________________|\n");
        printf("\n");
        }

void prioritySort (struct Schedule sc[], int numP) {
        sc[MAX_CAP-1].priority = LIM;
        int i, smallest, count = 0, time, burst[MAX_CAP];
        char porder[MAX_CAP][3]; int ptime[MAX_CAP], pcount = 0, save = MAX_CAP-1;

        for (i=0; i<numP; i++)
                burst[i] = sc[i].btime;

        for (time=0; count!=numP; time++) {
                smallest = MAX_CAP-1;
                for (i=0; i<numP; i++) {
                        if (sc[i].atime<time && sc[i].priority<sc[smallest].priority &&
sc[i].btime>0) {
                                smallest = i;
                                }
                        }
                sc[smallest].btime--;

                if (strcmp(sc[smallest].pID,sc[save].pID) != 0) {
                                        strcpy(porder[pcount],sc[smallest].pID);
```

```
                                        ptime[pcount] = time;
                                        pcount++;
                                        }
                save = smallest;

                if (sc[smallest].btime == 0) {
                        count++;
                        sc[smallest].wtime = time - sc[smallest].atime - burst[smallest];
                        sc[smallest].ttime = time - sc[smallest].atime;
                        }
                }

        for (i=0; i<numP; i++)
                sc[i].btime = burst[i];

        //print Gantt chart;
        printf("\nGantt chart:\n");
        printShortLine(pcount);
        printf("| |");
        for (int i=0; i<pcount; i++) {
                printf("  %s  | |",porder[i]);
                }
        printf("\n");
        printShortLine(pcount);
        printf(" 0 ");
        for (int i=0; i<pcount; i++) {
                printf("       %d",ptime[i]);
                }
        printf("\n\n");

        }

void roundrobin (struct Schedule sc[], int numP, int time_quantum) {
        int i, count, time, remain = numP, flag = 0, rt[MAX_CAP];
        printf("here\n");

        for (i=0; i<numP; i++) {
                rt[i] = sc[i].btime;
                }

        for (time=0, count=0; remain!=0; ) {
                printf("infor\n");
                if (rt[count]<=time_quantum && rt[count]>0) {
                        printf("inif\n");
                        time += rt[count];
                        rt[count] = 0;
                        flag = 1;
                        }
                else if (rt[count]>0) {
```

```
                printf("inelseif\n");
                rt[count] -= time_quantum;
                time += time_quantum;
                }
        else
                printf("cannot\n");

        if (rt[count] && flag==1) {
                printf("inifflag\n");
                remain--;
                sc[count].wtime = time - sc[count].atime - sc[count].btime;
                sc[count].ttime = time - sc[count].atime;
                flag = 0;
                }

        if (count == numP-1) {
                printf("inctif\n");
                count = 0;
                }
        else if (sc[count+1].atime <= time) {
                printf("inctelseif\n");
                count++;
                }
        else {
                printf("inctelse\n");
                count = 0;
                }
        printf("therefor\n");
        }

    }

void printLine() {
    printf(" _____\n");
    }

void printDashLine() {
    printf(" ----------\n");
    }

void printShortLine (int n) {
    for (int i=0; i<n; i++)
        printf("----------");
    printf("\n");
    }
```

Output:

Round robin:

```
kri@kri-ubuntu:~/workspace$ gcc -o prr cpu_pr_rr.c
kri@kri-ubuntu:~/workspace$ ./prr

                CPU SCHEDULING ALGORITHMS

_____MENU_____

1. ROUND ROBIN
2. PRIORITY


Enter choice: 1

__ROUND ROBIN__
Enter no. of processes: 5
Enter time quantum: 5

Enter processID: p1
Enter arrival time: 0
Enter burst time: 10

Enter processID: p2
Enter arrival time: 0
Enter burst time: 1

Enter processID: p3
Enter arrival time: 0
Enter burst time: 2

Enter processID: p4
Enter arrival time: 0
Enter burst time: 1

Enter processID: p5
Enter arrival time: 0
Enter burst time: 5

__No. of processes: 5__
 _____________________________________________
| pID   |A_time |B_time |W_time |T_time |R_time |
 ---------------------------------------------
| p2    | 0     | 1     | 5     | 6     | 5     |
| p3    | 0     | 2     | 6     | 8     | 6     |
| p4    | 0     | 1     | 8     | 9     | 8     |
| p5    | 0     | 5     | 9     | 14    | 9     |
| p1    | 0     | 10    | 9     | 19    | 0     |
|_____________________________________________|
|       Avg. waiting time: 7.40                |
|       Avg. turnaround time: 11.20            |
|       Avg. response time: 5.60               |
|_____________________________________________|


Gantt chart:
-------------------------------------------------------------
| |  p1  | |  p2  | |  p3  | |  p4  | |  p5  | |  p1  | |
-------------------------------------------------------------
 0        5       6       8       9       14       19
```

Priority:

```
Enter choice: 2

__PRIORITY__
Enter no. of processes: 3

Enter processID: p1
Enter arrival time: 0
Enter burst time: 5
Enter priority: 3

Enter processID: p2
Enter arrival time: 1
Enter burst time: 3
Enter priority: 1

Enter processID: p3
Enter arrival time: 2
Enter burst time: 2
Enter priority: 2
```

```
Gantt chart:
----------------------------------------
| |  p1  | |  p2  | |  p3  | |  p1  | |
----------------------------------------
 0        1       4       6       10


__No. of processes: 3__
 _____________________________________________
| pID   |A_time |B_time |Prty   |W_time |T_time |
 ---------------------------------------------
| p1    | 0     | 5     | 3     | 5     | 10    |
| p2    | 1     | 3     | 1     | 0     | 3     |
| p3    | 2     | 2     | 2     | 2     | 4     |
|_____________________________________________|
|       Avg. waiting time: 2.33               |
|       Avg. turnaround time: 5.67            |
|_____________________________________________|
```

Exiting:

```
Enter choice: 0
Exiting menu...
```

Learning outcomes:

- The various CPU Scheduling algorithms were understood.
- The CPU Scheduling algorithms, i.e., Round Robin and Priority were implemented in C.
- Preemptive scheduling was understood and implemented in C.

Assignment 5 – Interprocess communication

Date: 11/04/2022

Aim:

To develop the following applications that use interprocess communication concepts using shared memory:

1. An application for getting a name in parent and converting it into uppercase in child.
2. A client/server application for file transfer.
3. A client/server chat application.

Algorithm:

1) Application for converting to uppercase in child:

1. Start
2. Get the shared memory identifier.
3. Get the process id by forking.
4. If the process id is positive, then do the following:
    1. Attach a variable name to the shared memory.
    2. Get the name as input from the user.
    3. Detach the variable from the shared memory.
5. If the process id is 0, then do the following:
    1. Attach a variable to the shared memory.
    2. Convert the given name to uppercase.
    3. Print the name in uppercase.
    4. Detach the variable from the shared memory.
6. Remove the shared identifier and destroy the segment.
7. Stop

2) Application for file transfer:

Client side:

1. Start
2. Get the key for the shared memory by forking.
3. Get the shared memory identifier.
4. Attach a variable to the shared memory.
5. Get file name from the user.
6. Print the contents of the file onto the server.
7. Stop.

Server side:

1. Start
2. Get the key for the shared memory by forking.
3. Get the shared memory identifier.
4. Read the file name from the shared memory.
5. Read the contents of the file using a pointer.
6. Print the contents of the file.
7. Close the file.
8. Detach from the shared memory.
9. Destroy the shared memory space.
10. Stop

3) Chat application:

Client side:

1. Start
2. Get the process id and generate a shared memory identifier.
3. Set the key to some value.
4. Attach the structure to the shared memory.
5. Assign the second process id to the first and make the status ‘not ready’.
6. Signal the handler to receive a message.
7. Get a message from the user.
8. Set the status to ‘ready’.
9. Send the message by using the kill command to interrupt the process.
10. Wait until the status is ‘not ready’ and continue.
11. Detach the pointer from the shared memory.
12. Stop

Server side:

1. Start
2. Get the process id and assign the common key value to be the key.
3. Generate the shared memory identifier.
4. Attach the structure pointer to the shared memory.
5. Set the process id with the first process and the status to ‘not ready’.

6. Signal the handler function to receive a message.
7. Wait until the status is either ‘filled’ or ‘not ready’.
8. Get a message and set the status to ‘filled’.
9. Send the message by using the kill command to interrupt the process.
10. Detach the pointer from the shared memory.
11. Destroy the shared memory space.
12. Stop

Programs:

1) Application for converting to uppercase in child:

Code:

```
//Program to implement InterProcess Communication
#include <stdio.h>
#include <stdio_ext.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

//parent writes a char in shared memory which child reads and prints in upper case
void client(int id) {
        char *a;
        a = (char*) shmat(id,NULL,0);
        printf("\nChild -> Uppercase: ");
        for (int i=0; i<strlen(a); i++)
                printf("%c",toupper(a[i]));
        printf("\n");
        shmdt((void*)a);
        }

int main() {
        int pid, id;
        char *c;

        id = shmget(IPC_PRIVATE,1024,IPC_CREAT | 00666);
        c = (char*) shmat(id,NULL,0);

        printf("Enter string: ");
        scanf("%s",c);
```

```
pid = fork();

printf("Parent -> %s",c);

if (pid<0)
        printf("Error occurred");

else if (pid==0) {
        client(id);
        exit(0);
        }


wait(NULL);

printf("\n");

shmdt((void*)c);
shmctl(id,IPC_RMID,NULL);

return 0;
}
```

Output:

```
kri@kri-ubuntu:~/workspace$ gcc -o q51 ipc.c
kri@kri-ubuntu:~/workspace$ ./q51
Enter string: galaxy
Parent -> galaxy
Child -> Uppercase: GALAXY
Parent -> galaxy
kri@kri-ubuntu:~/workspace$
```

2) Application for file transfer:

Code: Server side

```
//Server for file transfer application
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/wait.h>
#include <ctype.h>
#include <signal.h>
#include <sys/types.h>
```

```
#include <sys/stat.h>
#include <fcntl.h>

struct memory{
   char file1[30];
   char file2[30];
   char content[200];
   int status,pid1,pid2;
         };

struct memory* shmptr;

void handler(int signum) {
   if(signum==SIGUSR1){
      printf("Finding file: %s\n",shmptr->file1);

      int fd = open(shmptr->file1, O_RDONLY);


      char content[200];
      read(fd,content,200);

      strcpy(shmptr->content,content);

      shmptr->status =0;

      close(fd);

      kill(shmptr->pid2,SIGUSR2);
      exit(0);
         }
         }

int main() {

   int pid = getpid();

   int id=shmget(111,sizeof(struct memory),IPC_CREAT | 0666);
   shmptr = (struct memory*)shmat(id, NULL, 0);
   shmptr->pid1=pid;
   shmptr->status=0;

   signal(SIGUSR1, handler);

   while (1) {
      sleep(1);
         }

   shmdt((void*)shmptr);
```

```
    shmctl(id, IPC_RMID, NULL);
    return 0;
        }
```

Code: Client side

```
//Client for file transfer application
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/wait.h>
#include <ctype.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

struct memory {
    char file1[30];
    char file2[30];
    char content[200];
    int status,pid1,pid2;
        };

struct memory* shmptr;

void handler(int signum) {
    //printf("interrupted\n");
    if(signum==SIGUSR2){
        int fd = open(shmptr->file2,O_WRONLY | O_CREAT,0644);
        write(fd,shmptr->content, strlen(shmptr->content));
        printf("%s saved as %s\n",shmptr->file1,shmptr->file2);
        exit(0);
            }
            }


int main() {
    int pid = getpid();

    int id=shmget(111,sizeof(struct memory),IPC_CREAT | 0666);
    shmptr = (struct memory*)shmat(id, NULL, 0);
    shmptr->pid2=pid;
    shmptr->status=0;

    char file1[30];
```

```
printf("File name: ");
scanf("%s",file1);
strcpy(shmptr->file1,file1);

char file2[30];
printf("Save file as: ");
scanf("%s",file2);
strcpy(shmptr->file2,file2);

shmptr->status=1;
kill(shmptr->pid1,SIGUSR1);

signal(SIGUSR2, handler);

while (1) {
   sleep(1);
      }

shmdt((void*)shmptr);
shmctl(id, IPC_RMID, NULL);
      }
```

Output:

```
kri@kri-ubuntu:~/workspace$ gcc -o q52c q2cl.c
kri@kri-ubuntu:~/workspace$ ./q52c
File name: foo.txt
Save file as: save.txt
foo.txt saved as save.txt
kri@kri-ubuntu:~/workspace$ cat save.txt
This is the first line.
This is the second line.
.
.
.
This is the last line.
kri@kri-ubuntu:~/workspace$
```

```
kri@kri-ubuntu:~/workspace$ gcc -o q52s q2server.c
kri@kri-ubuntu:~/workspace$ ./q52s
Finding file: foo.txt
kri@kri-ubuntu:~/workspace$ cat foo.txt
This is the first line.
This is the second line.
.
.
.
This is the last line.
kri@kri-ubuntu:~/workspace$
```

3) Chat application:

Code: Server side

```
//Server for chat application
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/wait.h>
#include <ctype.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

struct memory{
   char content[200];
   int status,pid1,pid2;
         };
struct memory* shmptr;

void handler(int signum) {
   if (signum==SIGUSR1){
      printf("%32s",shmptr->content);
         }
         }

int main() {
   int pid=getpid();

   int shmid=shmget(111,sizeof(struct memory),IPC_CREAT|0666);

   shmptr=(struct memory *)shmat(shmid,NULL,0);

   shmptr->pid1=pid;
   shmptr->status=0;

   signal(SIGUSR1, handler);

   printf("\n-------------------------------\n");
   printf("\tCHAT APPLICATION\n");
   printf("-------------------------------\n");

   int ch = 1;
```

```
   do {
      fgets (shmptr->content, 200, stdin);
      if (strcmp(shmptr->content,"exit")==10) {
         ch = 0;
         break;
         }
      else
         kill(shmptr->pid2,SIGUSR2);
         } while (ch==1);

   shmdt((void*)shmptr);
   shmctl(shmid, IPC_RMID, NULL);
         }
```

Code: Client side

```
//Client for chat application
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/wait.h>
#include <ctype.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

struct memory {
   char content[200];
   int status,pid1,pid2;
         };
struct memory* shmptr;

void handler(int signum) {
   if (signum==SIGUSR2) {
      printf("%32s",shmptr->content);
         }
         }

int main() {

   int pid=getpid();

   int shmid=shmget(111,sizeof(struct memory),IPC_CREAT|0666);

   shmptr=(struct memory *)shmat(shmid,NULL,0);
```

```
shmptr->pid2=pid;
shmptr->status=0;

signal(SIGUSR2, handler);

printf("\n-------------------------------\n");
printf("\tCHAT APPLICATION\n");
printf("-------------------------------\n");

int ch = 1;

do {
   fgets (shmptr->content, 200, stdin);
   if (strcmp(shmptr->content,"exit")==10) {
      ch = 0;
      break;
      }
   else
      kill(shmptr->pid1,SIGUSR1);
      } while (ch==1);

shmdt((void*)shmptr);
shmctl(shmid, IPC_RMID, NULL);
      }
```

Output:

```
kri@kri-ubuntu:~/workspace$ gcc -o q53c q3cl.c
kri@kri-ubuntu:~/workspace$ ./q53c

-------------------------------
        CHAT APPLICATION
-------------------------------
Hi!
                          Hello!
This is user1.
                This is user2.
exit
kri@kri-ubuntu:~/workspace$
```

```
kri@kri-ubuntu:~/workspace$ gcc -o q53s q3server.c
kri@kri-ubuntu:~/workspace$ ./q53s

-------------------------------
        CHAT APPLICATION
-------------------------------
                            Hi!
Hello!
                This is user1.
This is user2.
exit
kri@kri-ubuntu:~/workspace$
```

Learning outcomes:

- Interprocess communication was understood.
- A simulation of the process of sending and receiving signals between a client and a server was implemented.
- The concept of shared memory was understood and applied.

Assignment 6 – Implementation of Producer-Consumer Problem using Semaphores

Date: 18/04/2022

Aim:

i) To write a C program to create a parent/child process to implement the producer/consumer problem using semaphores in the pthread library.

ii) To modify the program as separate client/server process programs to generate 'N' random numbers in producer, write them into the shared memory and execute a consumer process to read them from the shared memory and display them on the terminal.

Algorithm:

i) Producer-Consumer problem using semaphores:

1. Start
2. Create a shared memory for the buffer and semaphores - *empty*, *full* and *mutex*. Get the shared memory identifier. Initialize the semaphores with values 'buffer-size', 0 and 1 respectively.
3. Create a parent and a child process with the parent acting as the producer and the child as the consumer.
4. Get a string as input from the user and fork the process.
5. In the producer process, produce an item by taking the first unvisited character from the string entered and placing it in the buffer array. Increment the semaphores *full* and *mutex*, and decrement the semaphore *empty* using the *wait* and *signal* operations suitably.
6. In the consumer process, consume an item by removing the first item in the buffer and displaying it on the terminal. Increment the semaphores *empty* and *mutex*, and decrement the semaphore *full* using the *wait* and *signal* operations appropriately.
7. Detach the pointer from the shared memory.
8. Destroy the shared memory space.
9. Stop

ii) Producer-Consumer problem with client/server process programs and random number generation:

Server side:

1. Start
2. Create a shared memory for the buffer and semaphores - *empty*, *full* and *mutex*. Get the shared memory identifier. Initialize the semaphores with values 'buffer-size', 0 and 1 respectively.
3. Get the number of numbers required as input from the user.
4. Randomly generate as many numbers as required and add them to the buffer one by one.
5. After every addition to the buffer, increment *full* and decrement *empty* using the wait and signal operations appropriately.
6. Detach the pointer from the shared memory.
7. Destroy the shared memory space.
8. Stop

Client side:

1. Start
2. Access the shared memory that contains the buffer.
3. Get the numbers stored in the buffer one by one and display them on the terminal.
4. Detach the pointer from the shared memory.
5. Stop

Programs:

1) Producer-Consumer problem using semaphores:

Code:

```
//Implementation of producer-consumer problem using semaphores

#include <stdio.h>
#include <semaphore.h>
#include <sys/shm.h>
#include <sys/sem.h>
#include <sys/wait.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/types.h>
#include <string.h>
#include <pthread.h>
#include <sys/ipc.h>

#define BUFSIZE 10
```

```
struct memory {
   char buffer[BUFSIZE];
   int count;
   sem_t full;
   sem_t empty;
   sem_t mutex;
        };
struct memory *shmptr;

char string[BUFSIZE];
int in_index=0;
int c=0;

void producer() {
        do {
                sem_wait(&(shmptr->empty));
                sem_wait(&(shmptr->mutex));

                shmptr->buffer[shmptr->count++]=string[in_index++];
                shmptr->buffer[shmptr->count]='\0';
                printf("Produced: %c\n",shmptr->buffer[shmptr->count-1]);
                printf("Buffer: %s\n",shmptr->buffer);
                sem_post(&(shmptr->mutex));
                sem_post(&(shmptr->full));

                sleep(1);
                } while(in_index<strlen(string));

        wait(NULL);
        printf("Producer operation completed!\n");
        exit(1);
        }

void consumer() {
            do {
               sem_wait(&(shmptr->full));
               sem_wait(&(shmptr->mutex));

               printf("Consumed: %c\n",shmptr->buffer[0]);
               memmove(shmptr->buffer,shmptr->buffer+1,strlen(shmptr->buffer));
               shmptr->count--;
               c++;

               printf("Buffer: %s\n",shmptr->buffer);
               sem_post(&(shmptr->mutex));
               sem_post(&(shmptr->empty));

               sleep(2);
                  } while(c<strlen(string));
```

```
        printf("Consumer operation completed!\n");
        exit(1);
    }

int main() {
        int shmid=shmget(IPC_PRIVATE,sizeof(struct memory),IPC_CREAT|0666);
        shmptr=(struct memory *)shmat(shmid,NULL,0);

        sem_init(&(shmptr->full),1,0);
        sem_init(&(shmptr->empty),1,BUFSIZE);
        sem_init(&(shmptr->mutex),1,1);
        shmptr->count=0;

        printf("Enter string: ");
        scanf("%s",string);

        int pid=fork();

        if(pid==-1)
           printf("Fork error\n");
        else if(pid==0)
           consumer();
        else
           producer();

        shmdt(shmptr);
        shmctl(shmid,IPC_RMID,NULL);
        sem_destroy(&(shmptr->empty));
        sem_destroy(&(shmptr->full));
        sem_destroy(&(shmptr->mutex));
        printf("Complete. Exiting...\n");
        }
```

Output:

```
kri@kri-ubuntu:~/workspace$ gcc -o q61 prod_cons.c -lpthread
kri@kri-ubuntu:~/workspace$ ./q61
Enter string: random
Produced: r
Buffer: r
Consumed: r
Buffer:
Produced: a
Buffer: a
Consumed: a
Buffer:
Produced: n
Buffer: n
Produced: d
Buffer: nd
Consumed: n
Buffer: d
Produced: o
Buffer: do
Produced: m
Buffer: dom
Consumed: d
Buffer: om
Consumed: o
Buffer: m
Consumed: m
Buffer:
Consumer operation completed!
```

2) Producer-Consumer problem with client/server process programs and random number generation:

Code: Server side

```
//Server side implementation of producer-consumer problem with random number generation

#include <stdio.h>
#include <semaphore.h>
#include <sys/shm.h>
#include <sys/sem.h>
#include <sys/wait.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/types.h>
#include <string.h>
#include <pthread.h>
#include <sys/ipc.h>
#include <time.h>

#define BUFSIZE 10

struct memory{
   int buffer[BUFSIZE];
   int count;
   sem_t full;
   sem_t empty;
   sem_t mutex;
   int n;
        };
struct memory *shmptr;

int main() {
            srand(time(0));
            int shmid=shmget(111,sizeof(struct memory),IPC_CREAT|0666);
            shmptr=(struct memory *)shmat(shmid,NULL,0);

            sem_init(&(shmptr->full),1,0);
            sem_init(&(shmptr->empty),1,BUFSIZE);
            sem_init(&(shmptr->mutex),1,1);
            shmptr->count=0;

            printf("No. of numbers: ");
            scanf("%d",&(shmptr->n));
```

```
int i=shmptr->n, rnum;
do {
        sem_wait(&(shmptr->empty));
        sem_wait(&(shmptr->mutex));

        rnum = rand()%100;
        shmptr->buffer[shmptr->count++]=rnum;
        printf("Produced: %d\n",shmptr->buffer[shmptr->count-1]);
        i--;

        sem_post(&(shmptr->mutex));
        sem_post(&(shmptr->full));
        sleep(1);
        } while(i>0);

printf("Producer operation completed!\n");

shmdt(shmptr);
shmctl(shmid,IPC_RMID,NULL);
sem_destroy(&(shmptr->empty));
sem_destroy(&(shmptr->full));
sem_destroy(&(shmptr->mutex));

printf("__Process completed__\n");
exit(1);
}
```

Code: Client side

```
//Client side implementation of producer-consumer problem with random number generation

#include <stdio.h>
#include <semaphore.h>
#include <sys/shm.h>
#include <sys/sem.h>
#include <sys/wait.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/types.h>
#include <string.h>
#include <pthread.h>
#include <sys/ipc.h>
#include <time.h>

#define BUFSIZE 10

struct memory{
        int buffer[BUFSIZE];
        int count;
        sem_t full;
```

```
        sem_t empty;
        sem_t mutex;
        int n;
        };
struct memory *shmptr;

int main() {
            srand(time(0));
            int shmid=shmget(111,sizeof(struct memory),IPC_CREAT|0666);
            shmptr=(struct memory *)shmat(shmid,NULL,0);

            int c=0;
            do {
                  sem_wait(&(shmptr->full));
                  sem_wait(&(shmptr->mutex));

                  printf("Consumed: %d\n",shmptr->buffer[0]);
                  memmove(shmptr->buffer,shmptr->buffer+1,sizeof(shmptr->buffer));
                  shmptr->count--;
                  c++;

                  sem_post(&(shmptr->mutex));
                  sem_post(&(shmptr->empty));
                  sleep(1);
                  } while(c<shmptr->n);

                  printf("\nConsumer operation completed!\n");

                  shmdt(shmptr);
                  shmctl(shmid,IPC_RMID,NULL);
                  sem_destroy(&(shmptr->empty));
                  sem_destroy(&(shmptr->full));
                  sem_destroy(&(shmptr->mutex));

                  printf("__Process complete. Exiting...__\n");
        exit(1);
        }
```

Output:

```
kri@kri-ubuntu:~/workspace$ gcc -o q62s sema_server.c -lpthread
kri@kri-ubuntu:~/workspace$ ./q62s
No. of numbers: 5
Produced: 89
Produced: 63
Produced: 94
Produced: 20
Produced: 5
Producer operation completed!
__Process completed__
kri@kri-ubuntu:~/workspace$
```

```
kri@kri-ubuntu:~/workspace$ gcc -o q62c sema_client.c -lpthread
kri@kri-ubuntu:~/workspace$ ./q62c
Consumed: 89
Consumed: 63
Consumed: 94
Consumed: 20
Consumed: 5

Consumer operation completed!
__Process complete. Exiting..__
kri@kri-ubuntu:~/workspace$
```

Learning outcomes:

- It was understood that semaphores can be used to solve various synchronization problems and can be implemented efficiently.
- The producer/consumer bounded buffer problem was understood by implementing it using semaphores.
- The method of handling semaphores between a server and a client was understood.

Assignment 7 – Implementation of Banker's Algorithm (Deadlock Avoidance) and Deadlock Detection

Date: 25/04/2022

Aim:

i) To develop a C program to implement Banker's algorithm for deadlock avoidance with multiple instances of resource types.

ii) To develop a C program to implement an algorithm for deadlock detection with multiple instances of resource types and display the processes involved in deadlock.

Algorithm:

i) Banker's Algorithm for Deadlock Avoidance:

1. Start
2. Read the following:
    1. No. of processes, *n*.
    2. No. of resources, *m*.
    3. Total number of instances of each resource.
    4. Maximum requirement of each resource.
    5. Allocated instances of each resource.
    6. Available number of instances of each resource.
3. Calculate the number of resources needed by each process by subtracting the allocated number of resources from the maximum number of resources.
4. Display the given details in a table.
5. To run the safety algorithm and check the system state:
    1. Let the vectors representing the work and finish status be vectors of lengths *m* and *n* respectively. Initialize:
        1. The work vector with the values of the available resources.
        2. The finish vector with 0 for each entry.
    2. Find a process such that its finish value is still 0 (i.e., the process has not been completed) and its need vector is less than the work vector.
    3. If such a process is found, add the resources allocated for the process to the work vector and set the finish states of the process to 1. Add the process to the safety sequence before repeating step 5.b.
    4. If the finish status of every process has been changed to 1, then the system is in a safe state. Print the safety sequence.

5. If one or more of the processes could not be completed, then display the processes for which the needed resources could not be allocated.

6. To put in a resource request, do the following:
    1. Get the process id that requests the resource and the request vector for the number of resources requested as input from the user.
    2. If the number of resources requested is lesser than the resources needed as well as the number of resources available, then:
        1. Update the available, needed and allocated vector resources for that process.
        2. Run the safety algorithm and print the safety sequence if it exists.
        3. If a safe sequence is obtained, grant the resources to the process by updating the system state. If not, inform the user that the resource request cannot be granted.
7. Stop

ii) Algorithm for Deadlock Detection:

1. Start
1. Read the following:
    1. No. of processes, *n*.
    2. No. of resources, *m*.
    3. Total number of instances of each resource.
    4. Maximum requirement of each resource.
    5. Allocated instances of each resource.
    6. Available number of instances of each resource.
2. Calculate the number of resources needed by each process by subtracting the allocated number of resources from the maximum number of resources.
3. Display the given details in a table.
4. To run the safety algorithm and check the system state:
    1. Let the vectors representing the work and finish status be vectors of lengths *m* and *n* respectively. Initialize:
        1. The work vector with the values of the available resources.
        2. The finish vector with 0 for each entry.
    2. Find a process such that its finish value is still 0 (i.e., the process has not been completed) and its need vector is less than the work vector.

3. If such a process is found, add the resources allocated for the process to the work vector and set the finish states of the process to 1. Add the process to the safety sequence before repeating step 5.b.
4. If the finish status of every process has been changed to 1, then the system is in a safe state. Print the safety sequence.
5. If one or more of the processes could not be completed, then display the incomplete processes that cause a potential deadlock.

5. Stop

Programs:

1) Banker's Algorithm for Deadlock Avoidance:

Code:

```
//Implementation of Banker's Algorithm for Deadlock Avoidance

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define P_CAP 10
#define R_CAP 5

void safety();

int main() {
	printf("\n__BANKER'S ALGORITHM__\n\n");

	int n, m, t=0;
	printf("Enter no. of processes: ");
	scanf("%d",&n);
	printf("Enter no. of resources: ");
	scanf("%d",&m);

	char pid[n][3], rid[m];
	int alloc[n][m], max[n][m], need[n][m], avail[n][m], total[m], available[m], work[m],
finish[n], safety[n], reqid, req[m];

	printf("\nEnter details of these resources:-\n\n");
	for (int j=0; j<m; j++) {
		printf("Resource name: ");
		scanf(" %c",&rid[j]);
		printf("\tTotal no. of instances: ");
		scanf("%d",&total[j]);
		}
```

```
printf("\n__MENU__\n1. Read data\n2. Print data\n3. Check system state\n4. Resource request\n5. Exit\n");

int choice = 1;
do {
	printf("\nEnter choice: ");
	scanf("%d",&choice);

	switch (choice) {
		case 1: {
			printf("\nEnter no. of available instances of each resource:-\n\n");
			for (int i=0; i<m; i++) {
				printf("Resource %c: ",rid[i]);
				scanf("%d",&avail[0][i]);
				work[i] = avail[0][i];
				available[i] = avail[0][i];
				}

			printf("\nEnter no. of processes: ");
			scanf("%d",&n);

			printf("Enter the details for each process:-\n\n");
			for (int i=0; i<n; i++) {
				printf("Process name: ");
				scanf("%s",pid[i]);
				printf("\tResources allocated: ");
				for (int j=0; j<m; j++) {
					scanf("%d",&alloc[i][j]);
					}
				printf("\tMaximum resources needed: ");
				for (int j=0; j<m; j++) {
					scanf("%d",&max[i][j]);
					}
				for (int j=0; j<m; j++) {
					need[i][j] = max[i][j] - alloc[i][j];
					}

				finish[i] = 0;
				}

			break;
			}

		case 2: {
			//print in table form
			printf("\nPID\tAllocated\tMaximum\t\tNeed\t\tAvailable\n");
			printf("\t");
			for (int col=1; col<=4; col++) {
				for (int j=0; j<m; j++)
```

```
                        printf("%c ",rid[j]);
                printf("\t\t");
                }
        printf("\n");

        for (int i=0; i<n; i++) {
                printf("%s\t",pid[i]);

                for (int j=0; j<m; j++)
                        printf("%d ",alloc[i][j]);
                printf("\t\t");

                for (int j=0; j<m; j++)
                        printf("%d ",max[i][j]);
                printf("\t\t");

                for (int j=0; j<m; j++)
                        printf("%d ",need[i][j]);
                printf("\t\t");

                if (i==0) {
                        for (int j=0; j<m; j++)
                                printf("%d ",available[j]);
                        }

                printf("\n");
                }
        break;
        }

case 4: {
        //resource request
        printf("Enter pid no. to request: P");
        scanf("%d",&reqid);

        printf("Resources requested: ");
        for (int j=0; j<m; j++)
                scanf("%d",&req[j]);

        for (int j=0; j<m; j++)
                work[j] = available[j];

        int status = 1;
        for (int j=0; j<m; j++)
                if (req[j] > need[reqid][j] || req[j] > work[j])
                        status = 0;
        if (status == 1) {
                for (int j=0; j<m; j++) {
                        work[j] = work[j] - req[j];
```

```
                        alloc[reqid][j] += req[j];
                        need[reqid][j] -= req[j];
                        avail[reqid][j] = work[j];
                        }
                }
        else {
                printf("The request cannot be satisfied with the current
resources.\n");
                break;
                }

        printf("Process %s:\n",pid[reqid]);
        printf("Available: ");
        for (int j=0; j<m; j++)
                printf("%d ",avail[reqid][j]);
        printf("\n");

        printf("Allocation: ");
        for (int j=0; j<m; j++)
                printf("%d ",alloc[reqid][j]);
        printf("\n");

        printf("Need: ");
        for (int j=0; j<m; j++)
                printf("%d ",need[reqid][j]);
        printf("\n");

        //print table
        printf("\nPID\tAllocated\tMaximum\t\tNeed\t\tAvailable\n");
        printf("\t");
        for (int col=1; col<=4; col++) {
                for (int j=0; j<m; j++)
                        printf("%c ",rid[j]);
                printf("\t\t");
                }
        printf("\n");
        for (int i=0; i<n; i++) {
                printf("%s\t",pid[i]);

                for (int j=0; j<m; j++)
                        printf("%d ",alloc[i][j]);
                printf("\t\t");

                for (int j=0; j<m; j++)
                        printf("%d ",max[i][j]);
                printf("\t\t");

                for (int j=0; j<m; j++)
                        printf("%d ",need[i][j]);
```

```
                printf("\t\t");

                if (i==0) {
                        for (int j=0; j<m; j++)
                                printf("%d ",work[j]);
                        }

                printf("\n");
                }
        }

case 3: {
        //display current status
        if (choice==3) {
                for (int i=0; i<n; i++) {
                        for (int j=0; j<m; j++) {
                                need[i][j] = max[i][j] - alloc[i][j];
                                }
                        }
                }

        for (int k=0; k<n; k++)
                finish[k] = 0;

        printf("\nChecking the status of the set of processes:- \n");

        int count = n, i = 0, flag = 0;
        while (count > 0) {
                if (finish[i] == 0) {
                        int status = 1;
                        for (int j=0; j<m; j++)
                                if (need[i][j] > work[j]) {
                                        status = 0;
                                        break;
                                        }
                        if (status == 1) {
                                flag = 1;
                                finish[i] = 1;
                                safety[n-count] = i;
                                count--;
                                for (int j=0; j<m; j++) {
                                        work[j] = work[j] + alloc[i][j];
                                        avail[i][j] = work[j];
                                        }
                                }

                        //printing details of the status of each process

printf("\nPID\tAllocated\t\tNeed\t\tStatus\t\tWork\n");
```

```
                                printf("%s\t",pid[i]);

                                for (int j=0; j<m; j++)
                                        printf("%d ",alloc[i][j]);
                                printf("\t\t\t");

                                for (int j=0; j<m; j++)
                                        printf("%d ",need[i][j]);
                                printf("\t-->   ");

                                if (status==1)
                                        printf("True\t\t");
                                else
                                        printf("False\t\t");

                                for (int j=0; j<m; j++)
                                        printf("%d ",work[j]);

                                printf("\n");
                                }

                        i++;

                        if (i==n) {
                                if (flag==0 && count!=0) {
                                        printf("\nResources could not be allocated
for the following processes: \n<");

                                        for (int k=0; k<n; k++)
                                                if (finish[k]==0)
                                                        printf("%s ",pid[k]);
                                        printf("\b>\n");
                                        if (choice==4) {
                                                for (int j=0; j<m; j++) {
                                                        alloc[reqid][j] -= req[j];
                                                        need[reqid][j] += req[j];
                                                        }
                                                }
                                        //printf("Possibility of deadlock!\n");
                                        break;
                                        }
                                else {
                                        i=0;
                                        flag=0;
                                        }
                                }
                        }

                if (count > 0) {
                        if (choice==4)
```

```
                                printf("Hence, the request cannot be granted.\n");
                        break;
                        }

                for (int j=0; j<m; j++)
                        work[j] = available[j];

                printf("\nSafety sequence:\n<");
                for (int i=0; i<n; i++)
                        printf("%s ",pid[safety[i]]);
                printf("\b>\n");

                if (choice==4)
                        printf("Process %s is thus granted.\n",pid[reqid]);

                break;
                }

        case 5: {
                printf("Exiting...\n");
                exit(0);
                break;
                }

        default: {
                printf("Invalid choice! Enter again...\n");
                }
        }
    } while (choice!=0);

return 0;
}
```

Output:

```
> ./a7avoid

__BANKER'S ALGORITHM__

Enter no. of processes: 5
Enter no. of resources: 3

Enter details of these resources:-

Resource name: A
    Total no. of instances: 10
Resource name: B
    Total no. of instances: 5
Resource name: C
    Total no. of instances: 7

__MENU__
1. Read data
2. Print data
3. Check system state
4. Resource request
5. Exit
```

```
Enter choice: 1

Enter no. of available instances of each resource:-

Resource A: 3
Resource B: 3
Resource C: 2

Enter no. of processes: 5
Enter the details for each process:-

Process name: P0
    Resources allocated: 0 1 0
    Maximum resources needed: 7 5 3
Process name: P1
    Resources allocated: 2 0 0
    Maximum resources needed: 3 2 2
Process name: P2
    Resources allocated: 3 0 2
    Maximum resources needed: 9 0 2
Process name: P3
    Resources allocated: 2 1 1
    Maximum resources needed: 2 2 2
Process name: P4
    Resources allocated: 0 0 2
    Maximum resources needed: 4 3 3
```

```
Enter choice: 2

PID Allocated   Maximum     Need        Available
    A B C       A B C       A B C       A B C
P0  0 1 0       7 5 3       7 4 3       3 3 2
P1  2 0 0       3 2 2       1 2 2
P2  3 0 2       9 0 2       6 0 0
P3  2 1 1       2 2 2       0 1 1
P4  0 0 2       4 3 3       4 3 1
```

```
Enter choice: 3

Checking the status of the set of processes:-

PID Allocated       Need        Status      Work
P0  0 1 0           7 4 3   -->   False     3 3 2

PID Allocated       Need        Status      Work
P1  2 0 0           1 2 2   -->   True      5 3 2

PID Allocated       Need        Status      Work
P2  3 0 2           6 0 0   -->   False     5 3 2

PID Allocated       Need        Status      Work
P3  2 1 1           0 1 1   -->   True      7 4 3

PID Allocated       Need        Status      Work
P4  0 0 2           4 3 1   -->   True      7 4 5

PID Allocated       Need        Status      Work
P0  0 1 0           7 4 3   -->   True      7 5 5

PID Allocated       Need        Status      Work
P2  3 0 2           6 0 0   -->   True      10 5 7

Safety sequence:
<P1 P3 P4 P0 P2>
```

```
Enter choice: 4
Enter pid no. to request: P1
Resources requested: 1 0 2
Process P1:
Available: 2 3 0
Allocation: 3 0 2
Need: 0 2 0

PID Allocated   Maximum     Need        Available
    A B C       A B C       A B C       A B C
P0  0 1 0       7 5 3       7 4 3       2 3 0
P1  3 0 2       3 2 2       0 2 0
P2  3 0 2       9 0 2       6 0 0
P3  2 1 1       2 2 2       0 1 1
P4  0 0 2       4 3 3       4 3 1
```

```
Checking the status of the set of processes:-

PID Allocated       Need        Status      Work
P0  0 1 0           7 4 3   -->   False     2 3 0

PID Allocated       Need        Status      Work
P1  3 0 2           0 2 0   -->   True      5 3 2

PID Allocated       Need        Status      Work
P2  3 0 2           6 0 0   -->   False     5 3 2

PID Allocated       Need        Status      Work
P3  2 1 1           0 1 1   -->   True      7 4 3

PID Allocated       Need        Status      Work
P4  0 0 2           4 3 1   -->   True      7 4 5

PID Allocated       Need        Status      Work
P0  0 1 0           7 4 3   -->   True      7 5 5

PID Allocated       Need        Status      Work
P2  3 0 2           6 0 0   -->   True      10 5 7

Safety sequence:
<P1 P3 P4 P0 P2>
Process P1 is thus granted.
```

```
Enter choice: 1

Enter no. of available instances of each resource:-

Resource A: 3
Resource B: 3
Resource C: 2

Enter no. of processes: 3
Enter the details for each process:-

Process name: P0
    Resources allocated: 0 1 0
    Maximum resources needed: 7 5 3
Process name: P1
    Resources allocated: 2 0 0
    Maximum resources needed: 3 2 2
Process name: P2
    Resources allocated: 3 0 2
    Maximum resources needed: 9 0 2

Enter choice: 2

PID Allocated   Maximum     Need        Available
    A B C       A B C       A B C       A B C
P0  0 1 0       7 5 3       7 4 3       3 3 2
P1  2 0 0       3 2 2       1 2 2
P2  3 0 2       9 0 2       6 0 0
```

```
Enter choice: 3

Checking the status of the set of processes:-

PID Allocated       Need        Status      Work
P0  0 1 0           7 4 3   -->   False     3 3 2

PID Allocated       Need        Status      Work
P1  2 0 0           1 2 2   -->   True      5 3 2

PID Allocated       Need        Status      Work
P2  3 0 2           6 0 0   -->   False     5 3 2

PID Allocated       Need        Status      Work
P0  0 1 0           7 4 3   -->   False     5 3 2

PID Allocated       Need        Status      Work
P2  3 0 2           6 0 0   -->   False     5 3 2

Resources could not be allocated for the following processes:
<P0 P2>

Enter choice: 4
Enter pid no. to request: P1
Resources requested: 1 0 2
Process P1:
Available: 2 3 0
Allocation: 3 0 2
Need: 0 2 0
```

```
PID Allocated   Maximum     Need        Available
    A B C       A B C       A B C       A B C
P0  0 1 0       7 5 3       7 4 3       2 3 0
P1  3 0 2       3 2 2       0 2 0
P2  3 0 2       9 0 2       6 0 0

Checking the status of the set of processes:-

PID Allocated       Need        Status      Work
P0  0 1 0           7 4 3   -->   False     2 3 0

PID Allocated       Need        Status      Work
P1  3 0 2           0 2 0   -->   True      5 3 2

PID Allocated       Need        Status      Work
P2  3 0 2           6 0 0   -->   False     5 3 2

PID Allocated       Need        Status      Work
P0  0 1 0           7 4 3   -->   False     5 3 2

PID Allocated       Need        Status      Work
P2  3 0 2           6 0 0   -->   False     5 3 2

Resources could not be allocated for the following processes:
<P0 P2>
Hence, the request cannot be granted.
```

```
Enter choice: 2

PID Allocated   Maximum     Need        Available
    A B C       A B C       A B C       A B C
P0  0 1 0       7 5 3       7 4 3       3 3 2
P1  2 0 0       3 2 2       1 2 2
P2  3 0 2       9 0 2       6 0 0

Enter choice: 4
Enter pid no. to request: P1
Resources requested: 2 2 2
The request cannot be satisfied with the current resources.
```

```
Enter choice: 4
Enter pid no. to request: P1
Resources requested: 0 1 0
Process P1:
Available: 3 2 2
Allocation: 2 1 0
Need: 1 1 2

PID Allocated   Maximum     Need        Available
    A B C       A B C       A B C       A B C
P0  0 1 0       7 5 3       7 4 3       3 2 2
P1  2 1 0       3 2 2       1 1 2
P2  3 0 2       9 0 2       6 0 0

Checking the status of the set of processes:-

PID Allocated       Need        Status      Work
P0  0 1 0           7 4 3   -->   False     3 2 2

PID Allocated       Need        Status      Work
P1  2 1 0           1 1 2   -->   True      5 3 2

PID Allocated       Need        Status      Work
P2  3 0 2           6 0 0   -->   False     5 3 2

PID Allocated       Need        Status      Work
P0  0 1 0           7 4 3   -->   False     5 3 2

PID Allocated       Need        Status      Work
P2  3 0 2           6 0 0   -->   False     5 3 2
```

```
Resources could not be allocated for the following processes:
<P0 P2>
Hence, the request cannot be granted.

Enter choice: 5
Exiting...
```

2) Algorithm for Deadlock Detection:

Code:

```
//Implementation of Algorithm for Deadlock Detection

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define P_CAP 10
#define R_CAP 5

void safety();

int main() {
	printf("\n__DEADLOCK DETECTION__\n\n");

	int n, m, t=0;
	printf("Enter no. of processes: ");
	scanf("%d",&n);
	printf("Enter no. of resources: ");
	scanf("%d",&m);

	char pid[n][3], rid[m];
	int alloc[n][m], max[n][m], need[n][m], avail[n][m], total[m], available[m], work[m],
finish[n], safety[n];

	printf("\nEnter details of these resources:-\n\n");
	for (int j=0; j<m; j++) {
		printf("Resource name: ");
		scanf(" %c",&rid[j]);
		printf("\tTotal no. of instances: ");
		scanf("%d",&total[j]);
		}

	printf("\n__MENU__\n1. Read data\n2. Print data\n3. Check system state\n4. Exit\n");

	int choice = 1;
	do {
		printf("\nEnter choice: ");
		scanf("%d",&choice);

		switch (choice) {
			case 1: {
				printf("\nEnter no. of available instances of each resource:-\n\n");
				for (int i=0; i<m; i++) {
					printf("Resource %c: ",rid[i]);
					scanf("%d",&avail[0][i]);
					work[i] = avail[0][i];
					available[i] = avail[0][i];
```

```
                }

        printf("\nEnter no. of processes: ");
        scanf("%d",&n);

        printf("Enter the details for each process:-\n\n");
        for (int i=0; i<n; i++) {
                printf("Process name: ");
                scanf("%s",pid[i]);
                printf("\tResources allocated: ");
                for (int j=0; j<m; j++) {
                        scanf("%d",&alloc[i][j]);
                        }
                printf("\tMaximum resources needed: ");
                for (int j=0; j<m; j++) {
                        scanf("%d",&max[i][j]);
                        }
                for (int j=0; j<m; j++) {
                        need[i][j] = max[i][j] - alloc[i][j];
                        }

                finish[i] = 0;
                }

        break;
        }

case 2: {
        //print in table form
        printf("\nPID\tAllocated\tMaximum\t\tNeed\t\tAvailable\n");
        printf("\t");
        for (int col=1; col<=4; col++) {
                for (int j=0; j<m; j++)
                        printf("%c ",rid[j]);
                printf("\t\t");
                }
        printf("\n");

        for (int i=0; i<n; i++) {
                printf("%s\t",pid[i]);

                for (int j=0; j<m; j++)
                        printf("%d ",alloc[i][j]);
                printf("\t\t");

                for (int j=0; j<m; j++)
                        printf("%d ",max[i][j]);
                printf("\t\t");
```

```
                for (int j=0; j<m; j++)
                        printf("%d ",need[i][j]);
                printf("\t\t");

                if (i==0) {
                        for (int j=0; j<m; j++)
                                printf("%d ",available[j]);
                        }

                printf("\n");
                }
        break;
        }

case 3: {
        //display current status
        if (choice==3) {
                for (int i=0; i<n; i++) {
                        for (int j=0; j<m; j++) {
                                need[i][j] = max[i][j] - alloc[i][j];
                                }
                        }
                }

        for (int k=0; k<n; k++)
                finish[k] = 0;

        printf("\nChecking the status of the set of processes:- \n");

        int count = n, i = 0, flag = 0;
        while (count > 0) {
                if (finish[i] == 0) {
                        int status = 1;
                        for (int j=0; j<m; j++)
                                if (need[i][j] > work[j]) {
                                        status = 0;
                                        break;
                                        }
                        if (status == 1) {
                                flag = 1;
                                finish[i] = 1;
                                safety[n-count] = i;
                                count--;
                                for (int j=0; j<m; j++) {
                                        work[j] = work[j] + alloc[i][j];
                                        avail[i][j] = work[j];
                                        }
                                }
```

```
                                        //printing details of the status of each process

printf("\nPID\tAllocated\t\tNeed\t\tStatus\t\tWork\n");
                                        printf("%s\t",pid[i]);

                                        for (int j=0; j<m; j++)
                                                printf("%d ",alloc[i][j]);
                                        printf("\t\t\t");

                                        for (int j=0; j<m; j++)
                                                printf("%d ",need[i][j]);
                                        printf("\t-->   ");

                                        if (status==1)
                                                printf("True\t\t");
                                        else
                                                printf("False\t\t");

                                        for (int j=0; j<m; j++)
                                                printf("%d ",work[j]);

                                        printf("\n");
                                        }

                                i++;

                                if (i==n) {
                                        if (flag==0 && count!=0) {
                                                printf("\nPossibility of deadlock!\n");
                                                printf("\nThe following processes may
cause the deadlock: \n<");
                                                for (int k=0; k<n; k++)
                                                        if (finish[k]==0)
                                                                printf("%s ",pid[k]);
                                                printf("\b>\n");
                                                break;
                                                }
                                        else {
                                                i=0;
                                                flag=0;
                                                }
                                        }
                                }

                        if (count > 0)
                                break;

                        for (int j=0; j<m; j++)
                                work[j] = available[j];
```

```
                        printf("\nSafety sequence:\n<");
                        for (int i=0; i<n; i++)
                                printf("%s ",pid[safety[i]]);
                        printf("\b>\n");

                        break;
                        }

                case 4: {
                        printf("Exiting...\n");
                        exit(0);
                        break;
                        }

                default: {
                        printf("Invalid choice! Enter again...\n");
                        }
                }
        } while (choice!=0);

return 0;
}
```

Output:

```
Enter choice: 1

Enter no. of available instances of each resource:-

Resource A: 3
Resource B: 3
Resource C: 2

Enter no. of processes: 5
Enter the details for each process:-

Process name: P0
    Resources allocated: 0 1 0
    Maximum resources needed: 7 5 3
Process name: P1
    Resources allocated: 2 0 0
    Maximum resources needed: 3 2 2
Process name: P2
    Resources allocated: 3 0 2
    Maximum resources needed: 9 0 2
Process name: P3
    Resources allocated: 2 1 1
    Maximum resources needed: 2 2 2
Process name: P4
    Resources allocated: 0 0 2
    Maximum resources needed: 4 3 3
```

```
./a7detect

__DEADLOCK DETECTION__

Enter no. of processes: 5
Enter no. of resources: 3

Enter details of these resources:-

Resource name: A
    Total no. of instances: 10
Resource name: B
    Total no. of instances: 5
Resource name: C
    Total no. of instances: 7

__MENU__
1. Read data
2. Print data
3. Check system state
4. Exit
```

```
Enter choice: 2

PID Allocated   Maximum     Need        Available
    A B C       A B C       A B C       A B C
P0  0 1 0       7 5 3       7 4 3       3 3 2
P1  2 0 0       3 2 2       1 2 2
P2  3 0 2       9 0 2       6 0 0
P3  2 1 1       2 2 2       0 1 1
P4  0 0 2       4 3 3       4 3 1
```

```
Enter choice: 3

Checking the status of the set of processes:-

PID Allocated       Need        Status      Work
P0  0 1 0           7 4 3   -->   False     3 3 2

PID Allocated       Need        Status      Work
P1  2 0 0           1 2 2   -->   True      5 3 2

PID Allocated       Need        Status      Work
P2  3 0 2           6 0 0   -->   False     5 3 2

PID Allocated       Need        Status      Work
P3  2 1 1           0 1 1   -->   True      7 4 3

PID Allocated       Need        Status      Work
P4  0 0 2           4 3 1   -->   True      7 4 5

PID Allocated       Need        Status      Work
P0  0 1 0           7 4 3   -->   True      7 5 5

PID Allocated       Need        Status      Work
P2  3 0 2           6 0 0   -->   True      10 5 7

Safety sequence:
<P1 P3 P4 P0 P2>
```

```
Enter choice: 1

Enter no. of available instances of each resource:-

Resource A: 3
Resource B: 3
Resource C: 2

Enter no. of processes: 3
Enter the details for each process:-

Process name: P0
    Resources allocated: 0 1 0
    Maximum resources needed: 7 5 3
Process name: P1
    Resources allocated: 2 0 0
    Maximum resources needed: 3 2 2
Process name: P2
    Resources allocated: 3 0 2
    Maximum resources needed: 9 0 2

Enter choice: 2

PID Allocated   Maximum     Need        Available
    A B C       A B C       A B C       A B C
P0  0 1 0       7 5 3       7 4 3       3 3 2
P1  2 0 0       3 2 2       1 2 2
P2  3 0 2       9 0 2       6 0 0
```

```
Enter choice: 3

Checking the status of the set of processes:-

PID Allocated       Need        Status      Work
P0  0 1 0           7 4 3   -->   False     3 3 2

PID Allocated       Need        Status      Work
P1  2 0 0           1 2 2   -->   True      5 3 2

PID Allocated       Need        Status      Work
P2  3 0 2           6 0 0   -->   False     5 3 2

PID Allocated       Need        Status      Work
P0  0 1 0           7 4 3   -->   False     5 3 2

PID Allocated       Need        Status      Work
P2  3 0 2           6 0 0   -->   False     5 3 2

Possibility of deadlock!

The following processes may cause the deadlock:
<P0 P2>

Enter choice: 4
Exiting...
```

Learning outcomes:

- The Banker's Algorithm for deadlock avoidance was understood and implemented.
- A safety sequence for process synchronization was obtained for a set of given processes and available resources.
- Methods for avoiding and detecting deadlocks were implemented.

Assignment 8 – Implementation of Memory Allocation Techniques: First Fit, Best Fit and Worst Fit

Date: 23/05/2022

Aim:

To implement the following memory allocation techniques:

1. First fit
2. Best fit
3. Worst fit

Algorithm:

1. Start
1. Get the required details on the partitioning of the physical memory such as:
    1. The number of partitions
    2. Starting addresses
    3. Ending addresses
2. Calculate the size of each partition and the free space available in each partition.
3. Display a menu with options for the three memory allocation techniques:
    1. First fit
    2. Best fit
    3. Worst fit
    5. For each technique, display a menu with options for allocation, deallocation, displaying the memory and for coalescing of holes.
6. Stop

1. *First fit:*
    1. Start
    2. To allocate a process:
        1. Read the process ID and the size of the process as input from the user.
        2. Iterate through the partitions until a partition with enough free space to accommodate the process is found.
        3. Insert the process into the physical memory and update the allocated and free memory accordingly.
        4. If no free space is available, print an error message indicating the same.
    3. To deallocate a process:

1. Read the process ID of the process to be deallocated as input from the user.
2. If the process is found in the physical memory, remove the process and create a hole in the memory space that the process was occupying.
3. If the process is not found in the physical memory, print an error message indicating the same.

4. To merge holes:
    1. If a partition contains a hole and the partitions immediately following it also contain holes, then note the starting address of the partition and the ending address of the last partition to contain a hole in this sequence.
    2. Set the newly acquired starting address and ending address as the starting and ending of a single partition that replaces the entire sequence of partitions containing holes.
    3. Repeat this strategy for all the sequences of holes in the physical memory.
5. Stop

2. *Best fit:*
    1. Start
    2. To allocate a process:
        1. Read the process ID and the size of the process as input from the user.
        2. Iterate through the partitions until a partition with the minimum free space to accommodate the process is found.
        3. Insert the process into the physical memory and update the allocated and free memory accordingly.
        4. If no free space is available, print an error message indicating the same.
    3. To deallocate a process:
        1. Read the process ID of the process to be deallocated as input from the user.
        2. If the process is found in the physical memory, remove the process and create a hole in the memory space that the process was occupying.
        3. If the process is not found in the physical memory, print an error message indicating the same.
    4. To merge holes:

1. If a partition contains a hole and the partitions immediately following it also contain holes, then note the starting address of the partition and the ending address of the last partition to contain a hole in this sequence.
2. Set the newly acquired starting address and ending address as the starting and ending of a single partition that replaces the entire sequence of partitions containing holes.
3. Repeat this strategy for all the sequences of holes in the physical memory.

5. Stop

3. *Worst fit:*
    1. Start
    2. To allocate a process:
        1. Read the process ID and the size of the process as input from the user.
        2. Iterate through the partitions until a partition with the maximum free space to accommodate the process is found.
        3. Insert the process into the physical memory and update the allocated and free memory accordingly.
        4. If no free space is available, print an error message indicating the same.
    3. To deallocate a process:
        1. Read the process ID of the process to be deallocated as input from the user.
        2. If the process is found in the physical memory, remove the process and create a hole in the memory space that the process was occupying.
        3. If the process is not found in the physical memory, print an error message indicating the same.
    4. To merge holes:
        1. If a partition contains a hole and the partitions immediately following it also contain holes, then note the starting address of the partition and the ending address of the last partition to contain a hole in this sequence.
        2. Set the newly acquired starting address and ending address as the starting and ending of a single partition that replaces the entire sequence of partitions containing holes.

3. Repeat this strategy for all the sequences of holes in the physical memory.

5. Stop

Programs:

Code:

```
// Program to implement Memory Management Algorithms - First fit, Best fit and Worst fit

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX 10

typedef struct Memnode {
  int start, end, size, freesize;
        char status[3];
        } mem;

void reset (mem space1[MAX], mem space2[MAX], int n) {
        for (int i=0; i<n; i++) {
                space1[i].start = space2[i].start;
                space1[i].end = space2[i].end;
                space1[i].size = space2[i].size;
                space1[i].freesize = space2[i].freesize;
                strcpy(space1[i].status,space2[i].status);
                }
        }

void shiftl (mem space[MAX], int pos, int n) {
        for (int i=pos; i<n; i++) {
                space[i].start = space[i+1].start;
                space[i].end = space[i+1].end;
                space[i].size = space[i+1].size;
                space[i].freesize = space[i+1].freesize;
                strcpy(space[i].status,space[i+1].status);
                }
        }

void displayfree (mem space[MAX], int numP, int allocs) {
        int n = numP-allocs;
        printf("\n");
        if (n==0) {
                printf("NULL\n");
                return;
                }
```

```
	for (int i=0; i<n; i++)
		printf("----------------");
	printf("\n");
	for (int i=0; i<numP; i++) {
		if (strcmp(space[i].status,"H")==0)
			printf("|\t\t%s\t\t|",space[i].status);
		}
	printf("\n");
	for (int i=0; i<n; i++)
		printf("----------------");
	printf("\n");
	for (int i=0; i<numP; i++) {
		if (strcmp(space[i].status,"H")==0)
			printf(" %d\t\t %d",space[i].start,space[i].end);
		}
	printf("\n");
	}

void display (mem space[MAX], int n) {
	printf("\n");
	if (n==0) {
		printf("NULL\n");
		return;
		}
	for (int i=0; i<n; i++)
		printf("----------------");
	printf("\n");
	for (int i=0; i<n; i++) {
		printf("|\t\t%s\t\t|",space[i].status);
		}
	printf("\n");
	for (int i=0; i<n; i++)
		printf("----------------");
	printf("\n");
	for (int i=0; i<n; i++) {
		printf(" %d\t\t %d",space[i].start,space[i].end);
		}
	printf("\n");
	}

int main() {
	printf("\n__MEMORY MANAGEMENT ALGORITHMS__\n");

	mem ph[MAX], free[MAX], alloc[MAX], temp[MAX];

	int numP;
	printf("\nEnter the no. of partitions in memory: ");
	scanf("%d",&numP);
```

```
        for (int i=0; i<numP; i++) {
                printf("\nPartition %d: ",i+1);
                printf("\n\tEnter starting address: ");
                scanf("%d",&ph[i].start);
                printf("\tEnter ending address: ");
                scanf("%d",&ph[i].end);
                ph[i].size = ph[i].end - ph[i].start;
                ph[i].freesize = ph[i].size;
                strcpy(ph[i].status,"H");
                }

        reset(temp,ph,numP);
        //displaying physical memory
        printf("\nPhysical memory: \n");
        display(ph,numP);
        //displaying copy of physical memory
        /*printf("\nTemp memory: \n");
        display(temp,numP);*/

        int algoch;
        do {
                printf("\nMemory allocation: \n1. First Fit\n2. Best Fit\n3. Worst Fit\n4. Exit
from program\n");
                printf("Enter choice: ");
                scanf("%d",&algoch);

                switch(algoch) {
                        case 1: {
                                printf("\n__First Fit Memory Allocation Algorithm__\n");
                                reset(ph,temp,numP);
                                reset(free,ph,numP);
                                int acount = 0, pcount = 0, numPh = numP, de = 0;   //counting
allocation partitions in memory

                                int choice;
                                do {
                                        printf("\nMENU: \n1. Allocate\n2. Deallocate\n3.
Display\n4. Coalescing of holes\n5. Back to program\n");
                                        printf("Enter choice: ");
                                        scanf("%d",&choice);

                                        switch(choice) {
                                                case 1: {
                                                        printf("\nAllocating memory: \n");
                                                        char pid[3]; int psize;
                                                        printf("Enter process ID: ");
                                                        scanf("%s",pid);
                                                        printf("Enter process size: ");
                                                        scanf("%d",&psize);
```

```
                                        for (int i=0; i<numPh; i++) {
                                                if (psize <= free[i].freesize) {
                                                        alloc[acount].start =
free[i].start;
                                                        alloc[acount].end =
alloc[acount].start + psize;
                                                        alloc[acount].size = psize;
                                                        alloc[acount].freesize = 0;

strcpy(alloc[acount].status,pid);
                                                        free[i].start += psize;
                                                        free[i].size -= psize;
                                                        free[i].freesize -= psize;
                                                        ph[pcount].freesize -=
psize;
                                                        numPh++;
                                                        pcount = i+acount+de;
                                                        for (int j=numPh-1;
j>pcount; j--) {
                                                                ph[j].start =
ph[j-1].start;
                                                                ph[j].end =
ph[j-1].end;
                                                                ph[j].freesize =
ph[j-1].freesize;
strcpy(ph[j].status,ph[j-1].status);
                                                                }
                                                        ph[pcount+1].start =
free[i].start;
                                                        ph[pcount].end =
alloc[acount].end;
                                                        ph[pcount].start =
alloc[acount].start;
                                                        ph[pcount].freesize = 0;

strcpy(ph[pcount].status,alloc[acount].status);
                                                        acount++;
                                                        break;
                                                        }
                                                }
                                        break;
                                        }
                                case 2: {
                                        printf("\nDeallocating memory: \n");
```

```
                char pid[3]; int psize;
                printf("Enter process ID: ");
                scanf("%s",pid);

                int ploc = -1, floc, aloc = -1;
                for (int i=0; i<numPh; i++) {
                        if (strcmp(ph[i].status,pid)==0)
                                ploc = i;
                        }
                for (int i=0; i<numPh; i++) {
                        if (strcmp(alloc[i].status,pid)==0)
                                aloc = i;
                        }
                floc = ploc-acount+1;
                psize = alloc[aloc].size;

                free[floc].start -= psize;
                free[floc].freesize += psize;

                shiftl(alloc,aloc,acount);
                acount--;

                strcpy(ph[ploc].status,"H");
                ph[ploc].freesize = psize;

                de++;

                break;
                }

        case 3: {
                printf("\nDisplaying memory: \n");

                printf("\nAllocated memory: \n");
                display(alloc,acount);
                printf("\nFree memory: \n");
                displayfree(ph,numPh,acount);
                printf("\nPhysical memory: \n");
                display(ph,numPh);

                break;
                }

        case 4: {
                printf("\nMerged holes! \n");
                int start = 0, end = 0;
                for (int i=1; i<numPh; i++) {
                        if (strcmp(ph[i-1].status,"H")==0)
{
```

```
                                                        start = ph[i-1].start;
                                                        int j = i;
                                                        while
(strcmp(ph[j].status,"H")==0 && j<numPh) {

shiftl(ph,j-1,numPh-1);
                                                                numPh--;
                                                                }
                                                        end = ph[i].end;
                                                        ph[i-1].start = start;
                                                        ph[i-1].end = end;
                                                        }
                                                }

                                                        break;
                                                        }

                                                case 5: {
                                                        printf("\nReturning to main program...\n");
                                                        break;
                                                        }

                                                default: {
                                                        printf("\nInvalid choice! Try again.\n");
                                                        }
                                                }
                                        } while(choice!=5);

                                break;
                                }

                        case 2: {
                                printf("\n__Best Fit Memory Allocation Algorithm__\n");
                                reset(ph,temp,numP);
                                reset(free,ph,numP);
                                int acount = 0, pcount = 0, numPh = numP, de = 0;   //counting
allocation partitions in memory

                                int choice;
                                do {
                                        printf("\nMENU: \n1. Allocate\n2. Deallocate\n3.
Display\n4. Coalescing of holes\n5. Back to program\n");
                                        printf("Enter choice: ");
                                        scanf("%d",&choice);

                                        switch(choice) {
                                                case 1: {
                                                        printf("\nAllocating memory: \n");
                                                        char pid[3]; int psize;
```

```
                                                printf("Enter process ID: ");
                                                scanf("%s",pid);
                                                printf("Enter process size: ");
                                                scanf("%d",&psize);

                                                int min_idx = 0;
                                                for (int i=0; i<numP; i++) {
                                                        if (free[i].freesize <
free[min_idx].freesize && psize <= free[i].freesize)
                                                                min_idx = i;
                                                        }

                                                if (psize <= free[min_idx].freesize) {
                                                        alloc[acount].start =
free[min_idx].start;
                                                        alloc[acount].end =
alloc[acount].start + psize;
                                                        alloc[acount].size = psize;
                                                        alloc[acount].freesize = 0;

strcpy(alloc[acount].status,pid);
                                                        free[min_idx].start +=
psize;
                                                        free[min_idx].size -= psize;
                                                        free[min_idx].freesize -=
psize;
                                                        numPh++;
                                                        pcount =
min_idx+acount+de;
                                                        for (int j=numPh-1;
j>pcount; j--) {
                                                                ph[j].start =
ph[j-1].start;
                                                                ph[j].end =
ph[j-1].end;
                                                                ph[j].freesize =
ph[j-1].freesize;
strcpy(ph[j].status,ph[j-1].status);
                                                                }
                                                        ph[pcount+1].start =
free[min_idx].start;
                                                        ph[pcount].end =
alloc[acount].end;
                                                        ph[pcount].start =
alloc[acount].start;
                                                        ph[pcount].freesize = 0;

strcpy(ph[pcount].status,alloc[acount].status);
```

```
                        acount++;
                        break;
                        }

                break;
                }

        case 2: {
                printf("\nDeallocating memory: \n");
                char pid[3]; int psize;
                printf("Enter process ID: ");
                scanf("%s",pid);

                int ploc = -1, floc, aloc = -1;
                for (int i=0; i<numPh; i++) {
                        if (strcmp(ph[i].status,pid)==0)
                                ploc = i;
                        }
                for (int i=0; i<numPh; i++) {
                        if (strcmp(alloc[i].status,pid)==0)
                                aloc = i;
                        }
                floc = ploc-acount+1;
                psize = alloc[aloc].size;

                free[floc].start -= psize;
                free[floc].freesize += psize;

                shiftl(alloc,aloc,acount);
                acount--;

                strcpy(ph[ploc].status,"H");
                ph[ploc].freesize = psize;

                de++;

                break;
                }

        case 3: {
                printf("\nDisplaying memory: \n");

                printf("\nAllocated memory: \n");
                display(alloc,acount);
                printf("\nFree memory: \n");
                displayfree(ph,numPh,acount);
                printf("\nPhysical memory: \n");
                display(ph,numPh);
```

```
                                        break;
                                        }

                                case 4: {
                                        printf("\nMerged holes! \n");
                                        int start = 0, end = 0;
                                        for (int i=1; i<numPh; i++) {
                                                if (strcmp(ph[i-1].status,"H")==0)
{
                                                        start = ph[i-1].start;
                                                        int j = i;
                                                        while
(strcmp(ph[j].status,"H")==0 && j<numPh) {

shiftl(ph,j-1,numPh-1);
                                                                numPh--;
                                                                }
                                                        end = ph[i].end;
                                                        ph[i-1].start = start;
                                                        ph[i-1].end = end;
                                                        }
                                                }

                                        break;
                                        }

                                case 5: {
                                        printf("\nReturning to main program...\n");
                                        break;
                                        }

                                default: {
                                        printf("\nInvalid choice! Try again.\n");
                                        }
                                }
                        } while(choice!=5);

                break;
                }

        case 3: {
                printf("\n__Worst Fit Memory Allocation Algorithm__\n");
                reset(ph,temp,numP);
                reset(free,ph,numP);
                int acount = 0, pcount = 0, numPh = numP, de = 0;   //counting
allocation partitions in memory

                int choice;
```

```
                        do {
                                printf("\nMENU: \n1. Allocate\n2. Deallocate\n3.
Display\n4. Coalescing of holes\n5. Back to program\n");
                                printf("Enter choice: ");
                                scanf("%d",&choice);

                                switch(choice) {
                                        case 1: {
                                                printf("\nAllocating memory: \n");
                                                char pid[3]; int psize;
                                                printf("Enter process ID: ");
                                                scanf("%s",pid);
                                                printf("Enter process size: ");
                                                scanf("%d",&psize);

                                                int max_idx = 0;
                                                for (int i=0; i<numP; i++) {
                                                        if (free[i].freesize >
free[max_idx].freesize && psize <= free[i].freesize)
                                                                max_idx = i;
                                                        }

                                                if (psize <= free[max_idx].freesize) {
                                                        alloc[acount].start =
free[max_idx].start;

                                                        alloc[acount].end =
alloc[acount].start + psize;
                                                        alloc[acount].size = psize;
                                                        alloc[acount].freesize = 0;
                                                        strcpy(alloc[acount].status,pid);
                                                        free[max_idx].start += psize;
                                                        free[max_idx].size -= psize;
                                                        free[max_idx].freesize -= psize;
                                                        ph[max_idx].freesize -= psize;
                                                        numPh++;
                                                        pcount = max_idx+acount+de;
                                                        for (int j=numPh-1; j>pcount; j--) {
                                                                ph[j].start = ph[j-1].start;
                                                                ph[j].end = ph[j-1].end;
                                                                ph[j].freesize =
ph[j-1].freesize;

strcpy(ph[j].status,ph[j-1].status);
                                                                }
                                                        ph[pcount+1].start =
free[max_idx].start;
                                                        ph[pcount].end =
alloc[acount].end;
```

```
                                        ph[pcount].start =
alloc[acount].start;
                                        ph[pcount].freesize = 0;

strcpy(ph[pcount].status,alloc[acount].status);

                                        acount++;
                                        break;
                                        }

                                break;
                                }

                        case 2: {
                                printf("\nDeallocating memory: \n");
                                char pid[3]; int psize;
                                printf("Enter process ID: ");
                                scanf("%s",pid);

                                int ploc = -1, floc, aloc = -1;
                                for (int i=0; i<numPh; i++) {
                                        if (strcmp(ph[i].status,pid)==0)
                                                ploc = i;
                                        }
                                for (int i=0; i<numPh; i++) {
                                        if (strcmp(alloc[i].status,pid)==0)
                                                aloc = i;
                                        }
                                floc = ploc-acount+1;
                                psize = alloc[aloc].size;

                                free[floc].start -= psize;
                                free[floc].freesize += psize;

                                shiftl(alloc,aloc,acount);
                                acount--;

                                strcpy(ph[ploc].status,"H");
                                ph[ploc].freesize = psize;

                                de++;

                                break;
                                }

                        case 3: {
                                printf("\nDisplaying memory: \n");

                                printf("\nAllocated memory: \n");
```

```
                                        display(alloc,acount);
                                        printf("\nFree memory: \n");
                                        displayfree(ph,numPh,acount);
                                        printf("\nPhysical memory: \n");
                                        display(ph,numPh);

                                        break;
                                        }

                                case 4: {
                                        printf("\nMerged holes! \n");
                                        int start = 0, end = 0;
                                        for (int i=1; i<numPh; i++) {
                                                if (strcmp(ph[i-1].status,"H")==0)
{
                                                        start = ph[i-1].start;
                                                        int j = i;
                                                        while
(strcmp(ph[j].status,"H")==0 && j<numPh) {

shiftl(ph,j-1,numPh-1);
                                                                numPh--;
                                                                }
                                                        end = ph[i].end;
                                                        ph[i-1].start = start;
                                                        ph[i-1].end = end;
                                                        }
                                                }

                                        break;
                                        }

                                case 5: {
                                        printf("\nReturning to main program...\n");
                                        break;
                                        }

                                default: {
                                        printf("\nInvalid choice! Try again.\n");
                                        }
                                }
                        } while(choice!=5);

                break;
                }

        case 4: {
                printf("\nExiting the program...\n");
                exit(0);
```

```
                        break;
                        }

                default: {
                        printf("\nInvalid choice! Try again.\n");
                        }
                }

        } while (algoch != 4);

return 0;
}
```

Output:

```
~/OSL$ ./a8

__MEMORY MANAGEMENT ALGORITHMS__

Enter the no. of partitions in memory: 4

Partition 1:
    Enter starting address: 100
    Enter ending address: 110

Partition 2:
    Enter starting address: 110
    Enter ending address: 117

Partition 3:
    Enter starting address: 117
    Enter ending address: 120

Partition 4:
    Enter starting address: 120
    Enter ending address: 124

Physical memory:

-------------------------------------------------------------
|      H      ||      H      ||      H      ||      H      |
-------------------------------------------------------------
 100         110 110         117 117         120 120         124
```

```
Memory allocation:
1. First Fit
2. Best Fit
3. Worst Fit
4. Exit from program
Enter choice: 1

__First Fit Memory Allocation Algorithm__

MENU:
1. Allocate
2. Deallocate
3. Display
4. Coalescing of holes
5. Back to program
Enter choice: 1

Allocating memory:
Enter process ID: P1
Enter process size: 5

MENU:
1. Allocate
2. Deallocate
3. Display
4. Coalescing of holes
5. Back to program
Enter choice: 3

Displaying memory:

Allocated memory:

---------------
|      P1     |
---------------
 100        105

Free memory:

-----------------------------------------------------------------
|      H      ||      H      ||      H      ||      H      |
-----------------------------------------------------------------
 105        110 110        117 117        120 120        124

Physical memory:

-------------------------------------------------------------------------------
|      P1     ||      H      ||      H      ||      H      ||      H      |
-------------------------------------------------------------------------------
 100        105 105        110 110        117 117        120 120        124

MENU:
1. Allocate
2. Deallocate
3. Display
4. Coalescing of holes
5. Back to program
Enter choice: 1
```

```
Allocating memory:
Enter process ID: P2
Enter process size: 6

MENU:
1. Allocate
2. Deallocate
3. Display
4. Coalescing of holes
5. Back to program
Enter choice: 3

Displaying memory:

Allocated memory:

--------------------------------
|       P1      ||      P2      |
--------------------------------
 100         105 110         116

Free memory:

-----------------------------------------------------------------
|       H       ||      H       ||      H       ||      H       |
-----------------------------------------------------------------
 105         110 116         117 117         120 120         124

Physical memory:

-----------------------------------------------------------------------------------------------
|       P1      ||      H       ||      P2      ||      H       ||      H       ||      H       |
-----------------------------------------------------------------------------------------------
 100         105 105         110 110         116 116         117 117         120 120         124

MENU:
1. Allocate
2. Deallocate
3. Display
4. Coalescing of holes
5. Back to program
Enter choice: 2

Deallocating memory:
Enter process ID: P2

MENU:
1. Allocate
2. Deallocate
3. Display
4. Coalescing of holes
5. Back to program
Enter choice: 3

Displaying memory:

Allocated memory:
```

```
Displaying memory:

Allocated memory:

---------------
|       P1      |
---------------
 100         105

Free memory:

-------------------------------------------------------------------------
|       H       ||      H       ||      H       ||      H       ||      H       |
-------------------------------------------------------------------------
 105         110 110         116 116         117 117         120 120         124

Physical memory:

-----------------------------------------------------------------------------------------
|       P1      ||      H       ||      H       ||      H       ||      H       ||      H       |
-----------------------------------------------------------------------------------------
 100         105 105         110 110         116 116         117 117         120 120         124

MENU:
1. Allocate
2. Deallocate
3. Display
4. Coalescing of holes
5. Back to program
Enter choice: 4
Merged holes!

MENU:
1. Allocate
2. Deallocate
3. Display
4. Coalescing of holes
5. Back to program
Enter choice: 3

Displaying memory:

Allocated memory:

---------------
|       P1      |
---------------
 100         105

Free memory:

---------------
|       H       |
---------------
 105         124
```

```
Physical memory:

---------------------------------
|       P1      ||      H       |
---------------------------------
 100         105 105         124

MENU:
1. Allocate
2. Deallocate
3. Display
4. Coalescing of holes
5. Back to program
Enter choice: 5

Returning to main program...

Memory allocation:
1. First Fit
2. Best Fit
3. Worst Fit
4. Exit from program
Enter choice: 2

__Best Fit Memory Allocation Algorithm__

MENU:
1. Allocate
2. Deallocate
3. Display
4. Coalescing of holes
5. Back to program
Enter choice: 1

Allocating memory:
Enter process ID: P1
Enter process size: 6

MENU:
1. Allocate
2. Deallocate
3. Display
4. Coalescing of holes
5. Back to program
Enter choice: 3

Displaying memory:

Allocated memory:

---------------
|       P1      |
---------------
 110         116

Free memory:

-----------------------------------------------------------------
|       H       ||      H       ||      H       ||      H       |
-----------------------------------------------------------------
Physical memory:

---------------------------------------------------------------------------------
|       H       ||      P1      ||      H       ||      H       ||      H       |
---------------------------------------------------------------------------------
 100         110 110         116 116         117 117         120 120         124

MENU:
1. Allocate
2. Deallocate
3. Display
4. Coalescing of holes
5. Back to program
Enter choice: 2

Deallocating memory:
Enter process ID: P1

MENU:
1. Allocate
2. Deallocate
3. Display
4. Coalescing of holes
5. Back to program
Enter choice: 4

Merged holes!

MENU:
1. Allocate
```

```
MENU:
1. Allocate
2. Deallocate
3. Display
4. Coalescing of holes
5. Back to program
Enter choice: 3

Displaying memory:

Allocated memory:

NULL

Free memory:

----------------
|       H       |
----------------
 100         124

Physical memory:

----------------
|       H       |
----------------
 100         124

MENU:
1. Allocate
2. Deallocate
3. Display
4. Coalescing of holes
5. Back to program
Enter choice: 3

Displaying memory:

Allocated memory:

NULL

Free memory:

----------------
|       H       |
----------------
 100         124

Physical memory:

----------------
|       H       |
----------------
 100         124

MENU:
1. Allocate
2. Deallocate
3. Display
4. Coalescing of holes
5. Back to program
Enter choice: 5

Returning to main program...

Memory allocation:
1. First Fit
2. Best Fit
3. Worst Fit
4. Exit from program
Enter choice: 3

__Worst Fit Memory Allocation Algorithm__

MENU:
1. Allocate
2. Deallocate
3. Display
4. Coalescing of holes
5. Back to program
Enter choice: 1

Allocating memory:
Enter process ID: P1
Enter process size: 2
```

```
MENU:
1. Allocate
2. Deallocate
3. Display
4. Coalescing of holes
5. Back to program
Enter choice: 3

Displaying memory:

Allocated memory:

---------------
|      P1     |
---------------
 100        102

Free memory:

-----------------------------------------------------------
|      H      ||      H      ||      H      ||      H      |
-----------------------------------------------------------
 102        110 110        117 117        120 120        124

Physical memory:

-------------------------------------------------------------------------------
|      P1     ||      H      ||      H      ||      H      ||      H      |
-------------------------------------------------------------------------------
 100        102 102        110 110        117 117        120 120        124
```

```
MENU:
1. Allocate
2. Deallocate
3. Display
4. Coalescing of holes
5. Back to program
Enter choice: 4

Merged holes!

MENU:
1. Allocate
2. Deallocate
3. Display
4. Coalescing of holes
5. Back to program
Enter choice: 2

Deallocating memory:
Enter process ID: P1

MENU:
1. Allocate
2. Deallocate
3. Display
4. Coalescing of holes
5. Back to program
Enter choice: 3

Displaying memory:

Allocated memory:

NULL

Free memory:

-------------------------------
|      H      ||      H      |
-------------------------------
 100        102 102        124

Physical memory:

-------------------------------
|      H      ||      H      |
-------------------------------
 100        102 102        124

MENU:
1. Allocate
2. Deallocate
3. Display
4. Coalescing of holes
5. Back to program
Enter choice: 5

Returning to main program...
```

```
Memory allocation:
1. First Fit
2. Best Fit
3. Worst Fit
4. Exit from program
Enter choice: 4

Exiting the program...
~/OSL$
```

Learning outcomes:

- The three memory allocation techniques - first fit, best fit and worst fit - were understood and implemented.
- Methods for allocating and deallocating processes were implemented.
- Availability of free space was increased by coalescing holes in partitions in the memory.

Assignment 9 – Implementation of Paging Technique

Date: 23/05/2022

Aim:

To develop a C program to implement the paging technique in memory management.

Algorithm:

1. Start
1. Define structures for frames and processes. Initialize the frames with a value that indicates allocated but unknown processes, say, -1.
2. Get the total size of the physical memory and the size of each page as input from the user.
3. Divide the physical memory into frames and compute the number of free frames available.
4. Display a menu with operations for allocation and deallocation of processes, displaying the page table and displaying the free frames available.
    1. Allocating a process:
        1. Read the entering process ID and size as input from the user.
        2. Compute the number of pages required for allocating this process.
        3. If the number of free frames is sufficient, iterate through all the frames to identify the free ones and assign pages to them accordingly.
        4. If the number of free frames is not sufficient, print a suitable error message.
    2. Deallocating a process:
        1. Read the required process ID as input from the user.
        2. If all the frames are free or the required process is not found, alert the user to the same.
        3. If not, iterate through all the frames to find the one with the required process.
        4. Append the released frames to the free frame queue and mark them as unallocated.
    3. Displaying the page table:
        1. For each process, iterate through the allocated frames and print all the page-frame pairs for the process.
    4. Displaying the free frames available:

1. Print the queue containing all the free frames.

5. Stop

Program:

Code:

```
//Program to implement paging technique
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct frames {
  int allocated;
  int process;
  int page;
} frames;

typedef struct process {
  int pid;
  int npages;
  int pages[50];
} process;

void init(frames *F, int n) {
  for (int i = 0; i < n; i++) {
    F[i].allocated = 1;
    F[i].process = -1; // allocated but unknown
  }
}

int main() {
  int phymemsize, pagesize, n_frames;
  int pid, psize;
  int np = 0;
  printf("\nPAGING TECHNIQUE\n");
  printf("\nEnter Physical Memory Size (in KB): ");
  scanf("%d", &phymemsize);
  printf("Enter Page Size (in KB): ");
  scanf("%d", &pagesize);
  n_frames = ceil(phymemsize / pagesize);
  frames F[n_frames];
  init(F, n_frames);
  int n, inputfree[50];
  printf("Enter number of free frames: ");
  scanf("%d", &n);
  printf("Enter string of free frames: ");
  for (int i = 0; i < n; i++) {
    scanf("%d", &inputfree[i]);
```

```
  F[inputfree[i] - 1].allocated = 0;
  F[inputfree[i] - 1].process = -1; // free
}
int choice;
process P[50];
do {
  printf("\nMENU: \n\t1.Process request\n\t2.Deallocation\n\t3.Page "
         "Table Display\n\t4.Free frame list display\n\t5.Exit\n");
  printf("Enter your choice: ");
  scanf("%d", &choice);
  switch (choice) {
  case 1:
    // process request
    printf("Enter process requirements\nID: P");
    scanf("%d", &pid);
    printf("Size (in KB): ");
    scanf("%d", &psize);
    int nreqpages = ceil(psize / pagesize);
    if (n > nreqpages) {
      int temp = nreqpages;
      int tempfp[50];

      for (int i = 0; i < n; i++) {
        tempfp[i] = inputfree[i];
      }

      P[np].pid = pid;
      P[np].npages = temp;
      printf("Process is divided into %d pages\n", nreqpages);
      //
      // while (temp > 0)
      for (int i = 0; i < n && temp > 0; i++) {
        // if (F[tempfp[i]-1].allocated == 0)
        //{
        printf("Page %d: Frame %d\n", nreqpages - temp, tempfp[i]);
        P[np].pages[nreqpages - temp] = tempfp[i];
        F[tempfp[i] - 1].allocated = 1;
        F[tempfp[i] - 1].process = pid;
        F[tempfp[i] - 1].page = nreqpages - temp;
        temp--;
        n--;
        // updating process
        P[np].pages[nreqpages - temp] = tempfp[i];
        for (int j = 0; j < n; j++)
          inputfree[j] = inputfree[j + 1];
        //}
      }
      np++;
    } else {
```

```
      printf(
          "Unsuccessful. Insufficient free frames available for process.\n");
    }
    break;
  case 2:
    // deallocation
    if (n == n_frames)
      printf("All blocks are free frames.\n");
    else {
      printf("Enter process ID to be deallocated: P");
      scanf("%d", &pid);
      for (int i = 0; i < np; i++) {
        if (P[i].pid == pid) {
          for (int j = 0; j < P[i].npages; j++) {
            inputfree[n] = P[i].pages[j];
            n += 1;
          }
        }
      }
      np--;
      printf("Deallocation successful\n");
    }
    break;
  case 3:
    // pagetable display
    printf("Page Table Display\n");
    for (int i = 0; i < np; i++) {
      printf("Process P%d\n", P[i].pid);
      for (int j = 0; j < P[i].npages; j++) {
        printf("Page %d: Frame %d\n", j, P[i].pages[j]);
      }
    }
    break;
  case 4:
    // free frame display
    printf("Free frames: ");
    for (int i = 0; i < n; i++)
      printf("%d ", inputfree[i]);
    printf("\n");
    break;
  case 5:
    exit(0);
  default:
    printf("Invalid option entered.\n");
    break;
  }
 } while (choice != 5);
 return 0;
}
```

Output:

```
root@hadoop-slave-3:~/krith# gcc -o pagn paging.c -lm
root@hadoop-slave-3:~/krith# ./pagn

PAGING TECHNIQUE

Enter Physical Memory Size (in KB): 32
Enter Page Size (in KB): 1
Enter number of free frames: 9
Enter string of free frames: 3 6 9 12 1 2 18 30 25

MENU:
        1.Process request
        2.Deallocation
        3.Page Table Display
        4.Free frame list display
        5.Exit
Enter your choice: 1
Enter process requirements
ID: P1
Size (in KB): 4
Process is divided into 4 pages
Page 0: Frame 3
Page 1: Frame 6
Page 2: Frame 9
Page 3: Frame 12

MENU:
        1.Process request
        2.Deallocation
        3.Page Table Display
        4.Free frame list display
        5.Exit
Enter your choice: 4
Free frames: 1 2 18 30 25

MENU:
        1.Process request
        2.Deallocation
        3.Page Table Display
        4.Free frame list display
        5.Exit
Enter your choice: 1
Enter process requirements
ID: P2
Size (in KB): 2
Process is divided into 2 pages
Page 0: Frame 1
Page 1: Frame 2

MENU:
        1.Process request
        2.Deallocation
        3.Page Table Display
        4.Free frame list display
        5.Exit
Enter your choice: 4
Free frames: 18 30 25
```

```
MENU:
        1.Process request
        2.Deallocation
        3.Page Table Display
        4.Free frame list display
        5.Exit
Enter your choice: 3
Page Table Display
Process P1
Page 0: Frame 3
Page 1: Frame 6
Page 2: Frame 9
Page 3: Frame 12
Process P2
Page 0: Frame 1
Page 1: Frame 2

MENU:
        1.Process request
        2.Deallocation
        3.Page Table Display
        4.Free frame list display
        5.Exit
Enter your choice: 2
Enter process ID to be deallocated: P1
Deallocation successful
MENU:
        1.Process request
        2.Deallocation
        3.Page Table Display
        4.Free frame list display
        5.Exit
Enter your choice: 4
Free frames: 18 30 25 3 6 9 12

MENU:
        1.Process request
        2.Deallocation
        3.Page Table Display
        4.Free frame list display
        5.Exit
Enter your choice: 5
root@hadoop-slave-3:~/krith#
```

Learning outcomes:

- Paging in memory management was understood.
- Paging techniques were implemented.
- The concepts of page tables and frames were understood and implemented.

Assignment 10 – Page Replacement Techniques:
FIFO, Optimal, LRU, LFU

Date: 23/05/2022

Aim:

To develop a C program to implement the page replacement algorithms (FIFO, Optimal, LRU and LFU) using linked list.

Algorithm:

1. Start
1. Read the number of frames.
2. Read the number of frames required by the process N.
3. Read the reference string for allocation of page frames.
4. Implement the chosen page replacement algorithm.
5. Stop

Page replacement algorithms:

1. *FIFO replacement:*
    1. Allocate the first N pages into the frames and increment the page faults accordingly.
    2. When the next frame in the reference string is not already available in the allocated list do:
        1. Look for the oldest one in the allocated frames and replace it with the next page frame.
        2. Increment the page fault whenever a frame is replaced.
    3. Display the allocated frame list after every replacement.

2. *Optimal replacement:*
    1. Allocate the first N pages into the frames and increment the page faults accordingly.
    2. When the next frame in the reference string is not already available in the allocated list do:
        1. Look for a frame in the reference string that will not be used for the longest period of time.

2. Increment the page fault whenever a frame is replaced. (Hint: Locate the position of each allocated frame in the reference string; identify a frame for replacement with the largest index position)
3. Display the allocated frame list after every replacement.

3. LRU replacement:

1. Allocate the first N pages into the frames and increment the page faults accordingly.
2. When the next frame in the reference string is not already available in the allocated list do:
    1. Look for a frame which has not been used recently.
    2. Increment the page fault whenever a frame is replaced.
3. Display the allocated frame list after every replacement.

4. LFU replacement:

1. Allocate the first N pages into the frames and increment the page faults accordingly.
2. When the next frame in the reference string is not already available in the allocated list do:
    1. Look for a frame which is least frequently used.
    2. Increment the page fault whenever a frame is replaced.
3. Display the allocated frame list after every replacement.

Code:

```
//Program to implement page replacement techniques
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int order= 0;
typedef struct Node {
  int d;
  struct Node *next;
  int freq;
  int order;
} node;

node *createlist() {
  node *head = (node *)malloc(sizeof(node));
```

```
  head->d = 0;
  head->next = NULL;
  head->freq = 0;
  return head;
}

void insertlast(node *head, int d) {
  node *ins = (node *)malloc(sizeof(node));
  ins->d = d;
  ins->freq = 1;

  node *temp = head;
  while (temp->next != NULL) {
    temp = temp->next;
  }
  ins->next = NULL;
  temp->next = ins;
}

void insertfirst(node *head, int d) {
  node *ins = (node *)malloc(sizeof(node));
  ins->d = d;
  ins->freq = 1;
  node *temp = head->next;
  ins->next = temp;
  head->next = ins;
}

int delete (node *prev) {
  int d;
  if (!prev)
    return -1;
  if (!prev->next)
    return -1;
  node *tmp = prev->next;
  d = tmp->d;
  prev->next = prev->next->next;
  free(tmp);
  return d;
}

int deletefirst(node *head) {
  int d;
  if (head->next == NULL) {
    printf("Empty List");
    return -1;
  }
  node *temp = head->next;
  if (temp->next != NULL) {
```

```
    head->next = temp->next;
    d = temp->d;
    free(temp);
  } else {
    d = temp->d;
    head->next = NULL;
  }
  return d;
}

int deletelast(node *head) {
  node *temp = head;
  if (head->next == NULL) {
    printf("Empty List!\n");
    return -1;
  }
  while (temp->next->next != NULL) {
    temp = temp->next;
  }
  delete (temp);
}

void display(node *head) {
  node *tmp = head->next;
  if (tmp == NULL) {
    printf("Empty!\n");
    return;
  }
  while (tmp) {
    printf(" %d", tmp->d);
    tmp = tmp->next;
  }
}

node *search(node *head, int d) {
  if (head->next == NULL)
    return NULL;

  node *tmp = head;
  while (tmp->next) {
    if (tmp->next->d == d)
      return tmp;
    tmp = tmp->next;
  }

  return NULL;
}

int frequency(int *seq, int d, int start, int end) {
```

```
  int itr = 0;
  if (start == end)
    return -1;
  for (int i = 0; i < start; i++)
  {
    if (seq[i] == d)
      itr++;
  }
  return itr;
}

int length(node *head) {
  node *tmp = head->next;
  if (tmp == NULL)
    return 0;

  int count = 0;
  while (tmp) {
    tmp = tmp->next;
    count++;
  }
  return count;
}

void putTable(int table[10][20], int n_frames, int n_updates) {
  printf("\n ");
  for (int i = 0; i < n_updates; i++)
    printf("+----");
  printf("+\n ");

  for (int i = 0; i < n_frames; i++) {
    for (int j = 0; j < n_updates; j++) {
      if (table[i][j] == -1)
        printf("| -  ");
      else
        printf("| %-2d ", table[i][j]);
    }
    printf("|\n ");
  }
  for (int i = 0; i < n_updates; i++)
    printf("+----");
  printf("+\n ");
}

void insertTable(node *tmp, int table[10][20], int n_frames, int faults) {
  for (int i = 0; i < n_frames; i++) {
    if (tmp) {
      table[i][faults] = tmp->d;
      tmp = tmp->next;
```

```
    } else
      table[i][faults] = -1;
  }
}

void FIFO(int seq[30], int len, int n_frames) {
  int faults = 0;
  int size = 0;
  int table[10][20];
  node *pg = createlist();
  node *oldest;
  printf("\n");
  printf("   Frame ->        In Memory      -> Faults \n\n");
  for (int i = 0; i < len; i++) {
    printf("     %-2d  ->", seq[i]);
    node *found = search(pg, seq[i]);
    node *tmp;
    if (!found) {
      if (size < n_frames) {
        insertlast(pg, seq[i]);
        size++;
        if (size == 1) {
          oldest = pg->next;
        }
      }
      else {
        oldest->d = seq[i];
        if (oldest->next) {
          oldest = oldest->next;
        } else {
          oldest = pg->next;
        }
      }
      insertTable(pg->next, table, n_frames, faults);
      faults++;
    }
    display(pg);
    // check formatting
    for (int i = length(pg) * 3; i <= 22; i++)
      printf(" ");
    printf("->    %-2d   \n", faults);
  }
  putTable(table, n_frames, faults);
}

void optimal(int seq[30], int len, int n_frames) {
  int size = 0;
  int faults = 0;
  int table[10][20];
```

```
int nextocc[n_frames];
node *pg = createlist();
printf("   Frame ->        In Memory      -> Faults \n\n");
for (int i = 0; i < len; i++) {
  printf("     %-2d  ->", seq[i]);
  node *found = search(pg, seq[i]);
  node *tmp;
  if (!found) {
    if (size < n_frames) {
      insertlast(pg, seq[i]);
      size++;
    }
    else {
      int distance = 0, maxd = 0, replace;
      int flag = 0;
      tmp = pg->next;
      node *change;
      while (tmp){
        flag = 0;
        for (int j = i; j<len && flag == 0; j++){
          if (seq[j] == tmp->d){
            flag = 1;
            distance = j-i;
            if (distance > maxd){
              maxd = distance;
              replace = seq[j];
            }
          }

          //printf("replace: %d ", replace);
        }
        if (flag == 0)
        {
          maxd = 99999;
          replace = tmp->d;
          break;
        }
        tmp = tmp->next;
      }
      change = search(pg, replace);
      change->next->d = seq[i];
    }
    insertTable(pg->next, table, n_frames, faults);
    faults++;
  }
  display(pg);
  // check formatting
  for (int i = length(pg) * 3; i <= 22; i++)
    printf(" ");
```

```
    printf("->    %-2d   \n", faults);
  }
  putTable(table, n_frames, faults);

}

void LRU(int seq[30], int len, int n_frames)
{
  int size = 0;
  int faults = 0;
  int table[10][20];
  int nextocc[n_frames];
  node *pg = createlist();
  printf("   Frame ->        In Memory      -> Faults \n\n");
  for (int i = 0; i < len; i++) {
    printf("     %-2d  ->", seq[i]);
    node *found = search(pg, seq[i]);
    node *tmp;
    if (!found) {
      if (size < n_frames) {
        insertlast(pg, seq[i]);
        size++;
      }
      else {
        int distance = 0, maxd = 0, replace;
        int flag = 0;
        tmp = pg->next;
        node *change;
        while (tmp){
          flag = 0;
          for (int j = i; j>0 && flag == 0; j--){
            if (seq[j] == tmp->d){
              flag = 1;
              distance = i-j;
              if (distance > maxd){
                maxd = distance;
                replace = seq[j];
              }
            }
          }
          if (flag == 0)
          {
            maxd = 99999;
            replace = tmp->d;
            break;
          }
          tmp = tmp->next;
        }
        change = search(pg, replace);
```

```
      change->next->d = seq[i];
     }
     insertTable(pg->next, table, n_frames, faults);
     faults++;
    }
    display(pg);
    // check formatting
    for (int i = length(pg) * 3; i <= 22; i++)
     printf(" ");
    printf("->    %-2d   \n", faults);
  }
  putTable(table, n_frames, faults);
}

void LFU(int seq[30], int len, int n_frames)
{
  int size = 0;
  int faults = 0;
  int table[10][20];
  int freq[n_frames];
  node *pg = createlist();
  printf("   Frame ->        In Memory      -> Faults \n\n");
  for (int i = 0; i < len; i++) {
    printf("     %-2d  ->", seq[i]);
    node *found = search(pg, seq[i]);
    node *tmp;
    if (!found) {
      if (size < n_frames) {
        insertlast(pg, seq[i]);
        size++;
        tmp = pg->next;
        while (tmp->next){
          tmp = tmp->next;
        }
        tmp->order = faults+1;
      }
      else {
        tmp = pg->next;
        int repl;
        int leastfreq = 999, eqno = 0;
        for( int j = 0; j < n_frames; j++)
          freq[j]=0;
        node *change;
        int k = 0;
        tmp = pg->next;
        while (tmp){
          freq[k] = frequency(seq, tmp->d, i, len);
          //printf("freq %d = %d\n", tmp->d, freq[k])
          if (freq[k]<leastfreq){
```

```
          leastfreq = freq[k];
          repl = tmp->d;
        }
        //leastfreq = tmp->d;
        k++;
        tmp = tmp->next;
      }
      tmp = pg->next;
      eqno = 0;
      for (int in = 0; in < k; in++){
        //printf("freq %d ", freq[in]);
        if (freq[in] == leastfreq)
          eqno++;
      }

      int minorder = 999;
      if (eqno > 1)
      {
        int in = 0;
        while (tmp){
          if (freq[in] == leastfreq)
            if (minorder>tmp->order){
              minorder = tmp->order;
              repl = tmp->d;
            }
          in++;
          tmp = tmp->next;
        }
        tmp = tmp->next;
      }
      change = search(pg, repl);
      change->next->d = seq[i];
      change->next->order = faults+1;
    }
    insertTable(pg->next, table, n_frames, faults);
    faults++;
  }
  display(pg);
  // check formatting
  for (int i = length(pg) * 3; i <= 22; i++)
    printf(" ");
  printf("->    %-2d   \n", faults);
 }
 putTable(table, n_frames, faults);
}


int main() {
 /*node* t1=createlist();
```

```
insertfirst(t1,1);
insertfirst(t1,2);
insertlast(t1,3);
deletefirst(t1);
deletelast(t1);
display(t1);  */
int n_free_frames = -1;
int n_reqd_frames = -1;
char buffer[20] = {0};
int sequence[30];
int choice = -1;
int len = 0;

while (1) {
  printf("\t\t\t\tPAGE REPLACEMENT TECHNIQUES\n");
  printf(" 1 - Read Input\n");
  printf(" 2 - FIFO\n");
  printf(" 3 - Optimal\n");
  printf(" 4 - LRU\n");
  printf(" 5 - LFU\n");
  printf(" 0 - Exit\n");
  printf(" -------------------------\n");
  printf("Enter your choice: ");
  scanf("%d", &choice);

  switch (choice) {
  case 0:
    exit(0);
  case 1:
    printf("Enter the number of free frames: ");
    scanf("%d", &n_free_frames);
    printf("Enter the number of required frames: ");
    scanf("%d", &n_reqd_frames);
    getchar();
    printf("Enter the length of Reference string: ");
    scanf("%d", &len);
    printf("Enter the Reference String: ");
    for (int i = 0; i < len; i++) {
      scanf("%d", &sequence[i]);
    }
    break;
  case 2:
    printf("\n\t\tFIFO\n");
    FIFO(sequence, len, n_reqd_frames);
    break;
  case 3:
    printf("\n\t\t\tOPTIMAL\n");
    optimal(sequence, len, n_reqd_frames);
    break;
```

```
    case 4:
      printf("\n\t\tLRU\n");
      LRU(sequence, len, n_reqd_frames);
      break;
    case 5:
      printf("\n\t\tLFU\n");
      LFU(sequence, len, n_reqd_frames);
      break;
    default:
      printf("Invalid Input!\n");
    }
    printf("\n");
  }
}
```

Output:

```
~/OSL$ ./a10
                PAGE REPLACEMENT TECHNIQUES
 1 - Read Input
 2 - FIFO
 3 - Optimal
 4 - LRU
 5 - LFU
 0 - Exit
 ------------------------
Enter your choice: 1
Enter the number of free frames: 3
Enter the number of required frames: 3
Enter the length of Reference string: 20
Enter the Reference String: 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
```

```
                PAGE REPLACEMENT TECHNIQUES
 1 - Read Input
 2 - FIFO
 3 - Optimal
 4 - LRU
 5 - LFU
 0 - Exit
 ------------------------
Enter your choice: 2
        FIFO

   Frame ->        In Memory       -> Faults

     7   -> 7                    ->    1
     0   -> 7 0                  ->    2
     1   -> 7 0 1                ->    3
     2   -> 2 0 1                ->    4
     0   -> 2 0 1                ->    4
     3   -> 2 3 1                ->    5
     0   -> 2 3 0                ->    6
     4   -> 4 3 0                ->    7
     2   -> 4 2 0                ->    8
     3   -> 4 2 3                ->    9
     0   -> 0 2 3                ->    10
     3   -> 0 2 3                ->    10
     2   -> 0 2 3                ->    10
     1   -> 0 1 3                ->    11
     2   -> 0 1 2                ->    12
     0   -> 0 1 2                ->    12
     1   -> 0 1 2                ->    12
     7   -> 7 1 2                ->    13
     0   -> 7 0 2                ->    14
     1   -> 7 0 1                ->    15
 +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
 | 7 | 7 | 7 | 2 | 2 | 2 | 4 | 4 | 4 | 0 | 0 | 0 | 7 | 7 | 7 |
 | - | 0 | 0 | 0 | 3 | 3 | 3 | 2 | 2 | 2 | 1 | 1 | 1 | 0 | 0 |
 | - | - | 1 | 1 | 1 | 0 | 0 | 0 | 3 | 3 | 3 | 2 | 2 | 2 | 1 |
 +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

```
                PAGE REPLACEMENT TECHNIQUES
 1 - Read Input
 2 - FIFO
 3 - Optimal
 4 - LRU
 5 - LFU
 0 - Exit
 ------------------------
Enter your choice: 3
```

```
          OPTIMAL
  Frame ->        In Memory      -> Faults

    7   -> 7                     ->    1
    0   -> 7 0                   ->    2
    1   -> 7 0 1                 ->    3
    2   -> 2 0 1                 ->    4
    0   -> 2 0 1                 ->    4
    3   -> 2 0 3                 ->    5
    0   -> 2 0 3                 ->    5
    4   -> 2 4 3                 ->    6
    2   -> 2 4 3                 ->    6
    3   -> 2 4 3                 ->    6
    0   -> 2 0 3                 ->    7
    3   -> 2 0 3                 ->    7
    2   -> 2 0 3                 ->    7
    1   -> 2 0 1                 ->    8
    2   -> 2 0 1                 ->    8
    0   -> 2 0 1                 ->    8
    1   -> 2 0 1                 ->    8
    7   -> 7 0 1                 ->    9
    0   -> 7 0 1                 ->    9
    1   -> 7 0 1                 ->    9
+----+----+----+----+----+----+----+----+----+
| 7  | 7  | 7  | 2  | 2  | 2  | 2  | 2  | 7  |
| -  | 0  | 0  | 0  | 0  | 4  | 0  | 0  | 0  |
| -  | -  | 1  | 1  | 3  | 3  | 3  | 1  | 1  |
+----+----+----+----+----+----+----+----+----+
```

```
                PAGE REPLACEMENT TECHNIQUES
 1 - Read Input
 2 - FIFO
 3 - Optimal
 4 - LRU
 5 - LFU
 0 - Exit
 -------------------------
Enter your choice: 4
        LRU
  Frame ->        In Memory      -> Faults

    7   -> 7                     ->    1
    0   -> 7 0                   ->    2
    1   -> 7 0 1                 ->    3
    2   -> 2 0 1                 ->    4
    0   -> 2 0 1                 ->    4
    3   -> 2 0 3                 ->    5
    0   -> 2 0 3                 ->    5
    4   -> 4 0 3                 ->    6
    2   -> 4 0 2                 ->    7
    3   -> 4 3 2                 ->    8
    0   -> 0 3 2                 ->    9
    3   -> 0 3 2                 ->    9
    2   -> 0 3 2                 ->    9
    1   -> 1 3 2                 ->    10
    2   -> 1 3 2                 ->    10
    0   -> 1 0 2                 ->    11
    1   -> 1 0 2                 ->    11
    7   -> 1 0 7                 ->    12
    0   -> 1 0 7                 ->    12
    1   -> 1 0 7                 ->    12

+----+----+----+----+----+----+----+----+----+----+----+----+
| 7  | 7  | 7  | 2  | 2  | 4  | 4  | 4  | 0  | 1  | 1  | 1  |
| -  | 0  | 0  | 0  | 0  | 0  | 0  | 3  | 3  | 3  | 0  | 0  |
| -  | -  | 1  | 1  | 3  | 3  | 2  | 2  | 2  | 2  | 2  | 7  |
+----+----+----+----+----+----+----+----+----+----+----+----+
```

```
              PAGE REPLACEMENT TECHNIQUES
 1 - Read Input
 2 - FIFO
 3 - Optimal
 4 - LRU
 5 - LFU
 0 - Exit
 ------------------------
Enter your choice: 5

       LFU
  Frame ->        In Memory      -> Faults

     7   -> 7                     ->    1
     0   -> 7 0                   ->    2
     1   -> 7 0 1                 ->    3
     2   -> 2 0 1                 ->    4
     0   -> 2 0 1                 ->    4
     3   -> 2 0 3                 ->    5
     0   -> 2 0 3                 ->    5
     4   -> 4 0 3                 ->    6
     2   -> 4 0 2                 ->    7
     3   -> 3 0 2                 ->    8
     0   -> 3 0 2                 ->    8
     3   -> 3 0 2                 ->    8
     2   -> 3 0 2                 ->    8
     1   -> 3 0 1                 ->    9
     2   -> 3 0 2                 ->    10
     0   -> 3 0 2                 ->    10
     1   -> 1 0 2                 ->    11
     7   -> 7 0 2                 ->    12
     0   -> 7 0 2                 ->    12
     1   -> 1 0 2                 ->    13
```

7	7	7	2	2	4	4	3	3	3	1	7	1
-	0	0	0	0	0	0	0	0	0	0	0	0
-	-	1	1	3	3	2	2	1	2	2	2	2

Learning outcomes:

- Page replacement techniques were understood and implemented.
- The different page replacement techniques were compared.
- The optimal page replacement technique was found to produce the least number of page faults.

Assignment 11 – Implementation of Threads

Date: 23/05/2022

Aim:

To create a multithreaded program that calculates various statistical values for a list of numbers passed as command line arguments.

Algorithm:

1. Start
1. Initialize the global variables sum, min, max and avg to zero.
2. Define functions to compute the minimum, maximum and average values of a given list of numbers.
3. In the main function, define the thread identifiers and initialize the thread attributes.
4. Read a list of elements from the command line arguments.
5. Create threads for each function.
6. Wait for the threads to close using the join function.
7. Print the results of the computations performed by each thread.
8. Stop
1.

Code:

```
//Program to implement threads
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>

int size;/* this data is shared by the thread(s) */
int arr[50];
float avgresult = 0, minresult = 0, maxresult = 0;

/* threads call this function */
void *avg()
{
  float sum = 0;
  for (int i = 0; i<size; i++)
    sum+=arr[i];
  avgresult = sum/size;
}

void *min()
{
  int minind;
```

```
  minind = 0;
  for (int i = 0 ; i<size; i++)
  {
    if (arr[minind] > arr[i])
      minind = i;
  }
  minresult = arr[minind];
}

void *max()
{
  int maxind;
  maxind = 0;
  for (int i = 0 ; i<size; i++)
  {
    if (arr[maxind] < arr[i])
      maxind = i;
  }
  maxresult = arr[maxind];
}

int main(int argc, char *argv[])
{
  pthread_t tid1; /* the thread identifier */
  pthread_attr_t attr1;
  pthread_t tid2; /* the thread identifier */
  pthread_attr_t attr2;
  pthread_t tid3; /* the thread identifier */
  pthread_attr_t attr3;
  pthread_attr_init(&attr1);
  pthread_attr_init(&attr2);
  pthread_attr_init(&attr3);
  //getting input
  size = argc - 1;
  for (int i = 0 ; i < size; i++)
    arr[i] = atoi(argv[i+1]);
  /* create the thread */
  pthread_create(&tid1,&attr1,avg,NULL);
  pthread_create(&tid2,&attr2,min,NULL);
  pthread_create(&tid3,&attr3,max,NULL);
  /* wait for the thread to exit */
  pthread_join(tid1,NULL);
  pthread_join(tid2,NULL);
  pthread_join(tid3,NULL);
  printf("\n");
  printf("The average value is %.2f\n", avgresult);
  printf("The minimum value is %.2f\n", minresult);
  printf("The maximum value is %.2f\n", maxresult);
}
```

Output:

```
~/OSL$ ./a11 90 81 78 95 79 72 85

The average value is 82.86
The minimum value is 72.00
The maximum value is 95.00
~/OSL$
```

Learning outcomes:

- Concurrent execution using threads was understood and implemented.
- Manipulation of threads using the pthread library was understood.

Assignment 12 – File Allocation Techniques: Sequential, Linked and Indexed Allocation

Date: 30/05/2022

Aim:

To develop a C program to implement the various file allocation techniques.

Algorithm:

1. Start
1. Get Main memory size and block size as input.
2. Create a Main memory with ‘n’ number of blocks of equal size.
3. Main memory is maintained as Linked List with structure containing block id, Free/Filename, Link to next Memory block , Link to Next File block (only for Linked Allocation), File block table (integer array to hold block numbers only for Indexed Allocation)
4. Get the number of files and their size as input.
5. Calculate the no. of blocks needed for each file.
6. Select the Allocation Algorithm – For every algorithm display Directory information and File information.
7. For Contiguous Allocation - For each file do the following:
    1. Generate a random number between 1 to ‘n’
    2. Check for a continuous number of needed file free blocks starting from that random block no.
    3. If free then allot that file in those continuous blocks and update the directory structure.
    4. Else, repeat step 1.
    5. If no continuous blocks are free then ‘no enough memory error’
    6. The Directory Structure should contain Filename, Starting Block, length (no. of blocks)
8. For Linked Allocation- For each file do the following:
    1. Generate a random number between 1 to ‘n’ blocks.
    2. Check if that block is free or not.
    3. If free then allot it for file. Repeat steps 1 to 3 for the needed number of blocks for the file and create a linked list in Main memory using the field “Link to Next File block”.

4. Update the Directory entry which contains Filename, Start block number, Ending Block Number.
5. Display the file blocks starting from start block number in Directory upto ending block number by traversing the Main memory Linked list using the field "Link to Next File block".

9. For Indexed Allocation - For each file do the following:
    1. Generate a random number between 1 to 'n' blocks for index block.
    2. Check if it is free. If not, repeat index block selection.
    3. Generate needed number of free blocks in random order for the file and store those block numbers in index block as array in File block table array.
    4. Display the Directory structure which contains the filename and index block number. Display the File Details by showing the index block number's File Block table.
10. Stop

Code:

```
//Linked list ADT for file allocation techniques
typedef Block Data;

typedef struct Node
{
   Data d;
   struct Node *next;
} Node;

typedef Node *List;

extern void init_block(Block *const);

List createEmptyList()
{
   Node *head = (Node *)malloc(sizeof(Node));
   init_block(&(head -> d));
   head->next = NULL;
   return head;
}

void insertLast(List head, const Data d)
{
   Node *new = (Node *)malloc(sizeof(Node));
   new->d = d;
   Node *tmp = head;

   while (tmp->next)
```

```
        tmp = tmp->next;

    new->next = NULL;
    tmp->next = new;
}

void insertFirst(List head, const Data d)
{
    Node *new = (Node *)malloc(sizeof(Node));
    new->d = d;

    new->next = head->next;
    head->next = new;
}

Data delete (List prev)
{
    Data rVal;
    if (!prev)
        return rVal;
    if (!prev->next)
        return rVal;

    Node *tmp = prev->next;
    rVal = tmp->d;
    prev->next = prev->next->next;
    free(tmp);

    return rVal;
}

Data deleteFirst(List head)
{
    Data rVal;
    if (head->next == NULL)
    {
        printf(" Empty List!\n");
        return rVal;
    }

    delete (head);
}

Data deleteLast(List head)
{
    Data rVal;
    if (head->next == NULL)
    {
        printf(" Empty List!\n");
```

```
        return rVal;
    }

    Node *tmp = head;
    while (tmp->next->next != NULL)
        tmp = tmp->next;

    delete (tmp);
}

void display(List head)
{
    Node *tmp = head->next;

    if (tmp == NULL)
    {
        printf(" Empty!\n");
        return;
    }

    while (tmp)
    {
        printf(" BID: %-2d\tStatus: %d\n", tmp->d.id, tmp->d.status);
        tmp = tmp->next;
    }
}

int length(List head)
{
    Node *tmp = head->next;
    if (tmp == NULL)
        return 0;

    int count = 0;
    while (tmp)
    {
        tmp = tmp->next;
        count++;
    }
    return count;
}

Node* search(List head, const int id)
{

    if (head->next == NULL)
        return NULL;

    Node *tmp = head -> next;
```

```
    while (tmp)
    {
        if (tmp->d.id == id)
            return tmp;
        tmp = tmp->next;
    }

    return NULL;
}

//Program to implement file allocation techniques
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

#define MAX 100
#define FREE 0

typedef struct File
{
    char name[21];
    int size;
    int start_block;
    int end_block;
    int *indices;
    int length;
} File;

void init_file(File *const);

typedef struct Directory
{
    File f[MAX];
    int size;
} Directory;

void init_dir(Directory *const);

typedef struct Block
{
    int id;
    unsigned status : 1;
    struct Block *next_file_blk;
} Block;

void init_block(Block *const);

#include "LinkedList.h"
```

```
void contiguous(File *const, const int, const int, const int);
void linked(File *const, const int, const int, const int);
void indexed(File *const, const int, const int, const int);

int main()
{
   printf("\n\t\t\tFILE ALLOCATION TECHNIQUES\n");

   int mem_size;
   int blk_size;
   int num_blks;
   int num_file;
   int choice;

   File f[MAX];

   printf(" Enter the size of memory: ");
   scanf("%d", &mem_size);
   printf(" Enter the size of block: ");
   scanf("%d", &blk_size);
   num_blks = mem_size / blk_size;

   printf(" Enter the number of files: ");
   scanf("%d", &num_file);
   getchar();

   for (int i = 0; i < num_file; i++)
   {
      printf(" Enter the name of file: ");
      scanf("%[^\n]", f[i].name);

      printf(" Enter the size of file: ");
      scanf("%d", &f[i].size);
      getchar();
   }

   while (1)
   {
      printf("\n\nMENU:\n");
      printf(" 1 - Contiguous\n");
      printf(" 2 - Linked\n");
      printf(" 3 - Indexed\n");
      printf(" 0 - Exit\n");
      printf(" ----------------------\n");
      printf(" Enter your choice: ");
      scanf("%d", &choice);

      switch (choice)
```

```
        {
        case 0:
            exit(0);
        case 1:
            contiguous(f, num_file, blk_size, num_blks);
            break;
        case 2:
            linked(f, num_file, blk_size, num_blks);
            break;
        case 3:
            indexed(f, num_file, blk_size, num_blks);
            break;

        default:
            printf(" Invalid Input!\n");
        }
    }
}

void init_file(File *const f)
{
    strcpy(f->name, "");
    f->start_block = -1;
    f->end_block = -1;
    f->size = -1;
    f->indices = NULL;
    f->length = -1;
}

void init_dir(Directory *const d)
{
    d->size = 0;
    for (int i = 0; i < MAX; i++)
        init_file(&(d->f[i]));
}

void init_block(Block *const b)
{
    b->status = FREE;
    b->id = -1;
    b->next_file_blk = NULL;
}

void contiguous(File *const f, const int n_files, const int blk_size, const int num_blk)
{
    List list = createEmptyList();

    Block b;
    init_block(&b);
```

```
Node *ptr, *tmp;

int blocks_visited, flag, id, counter, blk_req;
int start, end;

for (int i = 0; i < num_blk; i++)
{
   b.id = i;
   insertLast(list, b);
}

for (int i = 0; i < n_files; i++)
{
   blocks_visited = 0;
   flag = 0;
   blk_req = f[i].size / blk_size;
   if (f[i].size % blk_size)
      blk_req++;

   while (blocks_visited < num_blk && !flag)
   {
      id = random() % num_blk;
      ptr = search(list, id);
      if (ptr->d.status != FREE)
      {
         blocks_visited++;
         continue;
      }

      counter = 0;

      start = ptr->d.id;

      tmp = ptr;
      while (tmp)
      {
         if (tmp->d.status == FREE)
         {
            counter++;
            if (counter == blk_req)
            {
               flag = 1;
               break;
            }
         }
         else
            break;
         tmp = tmp->next;
```

```
            }

            if (flag)
            {
               f[i].start_block = start;
               f[i].length = blk_req;
               tmp = ptr;
               for (int i = 0; i < blk_req; i++)
               {
                  tmp->d.status = 1;
                  tmp = tmp->next;
               }
            }
            else
               blocks_visited++;
         }
         if (!flag)
            printf(" Unable to allocate file: %s\n!", f[i].name);
      }

      printf("\n\t\tDIRECTORY STRUCTURE\n");
      printf(" +----------------------+-------+--------+\n");
      printf(" |      File Name       | Start | Length |\n");
      printf(" +----------------------+-------+--------+\n");

      for (int i = 0; i < n_files; i++)
         if (f[i].length > 0)
            printf(" | %-20s | %-5d | %-6d |\n", f[i].name, f[i].start_block, f[i].length);
      printf(" +----------------------+-------+--------+\n");
   }
   void linked(File *const f, const int n_files, const int blk_size, const int num_blk)
   {
      List list = createEmptyList();

      Block b;
      init_block(&b);

      Node *ptr, *tmp, *left, *right;

      int blocks_visited, flag, id, counter, blk_req;

      for (int i = 0; i < num_blk; i++)
      {
         b.id = i;
         insertLast(list, b);
      }

      for (int i = 0; i < n_files; i++)
      {
```

```
counter = 0;
blocks_visited = 0;
flag = 0;
blk_req = f[i].size / blk_size;
if (f[i].size % blk_size)
   blk_req++;
int *allocated = (int *)calloc(blk_req, sizeof(int));

while (blocks_visited < num_blk && !flag)
{
   id = random() % num_blk;
   ptr = search(list, id);

   if (ptr->d.status != FREE)
   {
      blocks_visited++;
      continue;
   }
   ptr -> d.status = 1;
   allocated[counter++] = id;

   if (counter == blk_req)
      flag = 1;
}
if (!flag){
   printf(" Unable to allocate file: %s\n", f[i].name);
   for(int i = 0; i < counter; i++){
      ptr = search(list, allocated[i]);
      ptr -> d.status = FREE;
   }
   free(allocated);
}
else
{

   f[i].start_block = allocated[0];
   f[i].end_block = allocated[blk_req - 1];
   f[i].length = blk_req;
   for (int i = 0; i < blk_req - 1; i++)
   {

      left = search(list, allocated[i]);
      right = search(list, allocated[i + 1]);
      left->d.next_file_blk = &(right->d);
      left->d.status = 1;
   }
   right->d.next_file_blk = NULL;
   free(allocated);
}
```

```
    }
    printf("\n\t\tDIRECTORY STRUCTURE\n");
    printf(" +---------------------+------------+-----------+\n");
    printf(" |      File Name      | Start Block | End Block |\n");
    printf(" +---------------------+------------+-----------+\n");
    for (int i = 0; i < n_files; i++)
        if (f[i].end_block >= 0)
            printf(" | %-20s |     %-2d      |    %-2d     |\n",
                  f[i].name, f[i].start_block, f[i].end_block);
    printf(" +---------------------+------------+-----------+\n");

    printf("\n");
    for (int i = 0; i < n_files; i++)

        if (f[i].start_block >= 0)
        {
            printf("\n\n File Name: %s\n ",f[i].name);
            ptr = search(list, f[i].start_block);
            Block *b = &(ptr->d);
            while (b)
            {
                printf("%-2d ", b->id);
                b = b->next_file_blk;
            }
        }
}
void indexed(File *const f, const int n_files, const int blk_size, const int num_blk)
{
    List list = createEmptyList();

    Block b;
    init_block(&b);

    Node *ptr, *tmp;

    int blocks_visited, flag, id, counter, blk_req;
    int start, end;

    for (int i = 0; i < num_blk; i++) {
        b.id = i;
        insertLast(list, b);
    }

    for (int i = 0; i < n_files; i++) {
        blocks_visited = 0;
        flag = 0;
        blk_req = f[i].size / blk_size;
        if (f[i].size % blk_size)
            blk_req++;
```

```
        f[i].indices = (int *)calloc(blk_req + 1, sizeof(int));
        f[i].length = blk_req;
        counter = 0;
        while (blocks_visited < num_blk && !flag)
        {
            id = random() % num_blk;
            ptr = search(list, id);
            if (ptr->d.status == FREE)
            {
                f[i].indices[counter++] = id;
                if (counter == blk_req + 1)
                {
                    flag = 1;
                    break;
                }
            }
            else
                blocks_visited++;
        }
        if (!flag)
        {
            printf(" Unable to allocate memory for file: %s\n", f[i].name);
            free(f[i].indices);
            f[i].indices = NULL;
        }
    }
    printf("\n\t\tDIRECTORY STRUCTURE\n");
    printf(" +----------------------+-------------+\n");
    printf(" |      File Name       | Index Block |\n");
    printf(" +----------------------+-------------+\n");
    for (int i = 0; i < n_files; i++)
        if (f[i].indices)
            printf(" | %-20s |     %-2d      |\n", f[i].name, f[i].indices[0]);
    printf(" +----------------------+-------------+\n");

    printf("\n\n");
    printf(" +----------------------+----------------+\n");
    printf(" |      File Name       | Blocks Indexed |\n");
    printf(" +----------------------+----------------+\n");
    for (int i = 0; i < n_files; i++)
    {
        if (f[i].indices)
        {
            for (int j = 1; j <= f[i].length; j++)
                printf(" | %-20s |       %-2d       |\n", ((j > 1) ? "" : f[i].name), f[i].indices[j]);
        }
        printf(" +----------------------+----------------+\n");
    }
}
```

Output:

```
                        FILE ALLOCATION TECHNIQUES
Enter the size of memory: 100
Enter the size of block: 5
Enter the number of files: 3
Enter the name of file: f1
Enter the size of file: 15
Enter the name of file: f2
Enter the size of file: 20
Enter the name of file: f3
Enter the size of file: 10


MENU:
1 - Contiguous
2 - Linked
3 - Indexed
0 - Exit
----------------------
Enter your choice: 1

               DIRECTORY STRUCTURE
+----------------------+-------+--------+
|      File Name       | Start | Length |
+----------------------+-------+--------+
| f1                   | 3     | 3      |
| f2                   | 6     | 4      |
| f3                   | 17    | 2      |
+----------------------+-------+--------+


MENU:
1 - Contiguous
2 - Linked
3 - Indexed
0 - Exit
----------------------
```

```
Enter your choice: 2

               DIRECTORY STRUCTURE
+----------------------+-------------+-----------+
|      File Name       | Start Block | End Block |
+----------------------+-------------+-----------+
| f1                   |     15      |    6      |
| f2                   |     12      |    2      |
| f3                   |     7       |    10     |
+----------------------+-------------+-----------+



File Name: f1
15 13 6

File Name: f2
12 9  1  2

File Name: f3
7  10
```

```
MENU:
 1 - Contiguous
 2 - Linked
 3 - Indexed
 0 - Exit
 ----------------------
 Enter your choice: 3

                DIRECTORY STRUCTURE
 +----------------------+-------------+
 |      File Name       | Index Block |
 +----------------------+-------------+
 | f1                   |     19      |
 | f2                   |     6       |
 | f3                   |     7       |
 +----------------------+-------------+
 +----------------------+----------------+
 |      File Name       | Blocks Indexed |
 +----------------------+----------------+
 | f1                   |       3        |
 |                      |       6        |
 |                      |       0        |
 +----------------------+----------------+
 | f2                   |       12       |
 |                      |       16       |
 |                      |       11       |
 |                      |       8        |
 +----------------------+----------------+
 | f3                   |       9        |
 |                      |       2        |
 +----------------------+----------------+


MENU:
 1 - Contiguous
 2 - Linked
 3 - Indexed
 0 - Exit
 ----------------------
 Enter your choice: 0
root@hadoop-slave-3:~/krith#
```

Learning outcomes:

- File allocation techniques were understood and implemented.
- Files were allocated in the main memory according to the sequential, linked and indexed allocation techniques.

Assignment 13 – File Orgnaisation Techniques: Single Level and Hierarchical Directory Structures

Date: 30/05/2022

Aim:

To develop a C program to implement the following file organization techniques:
a) Single level Directory
b) Hierarchical Structure

Algorithm:

1. Start
1. Let the user choose between single level and hierarchical directory structures.
2. Single Level Directory
    a. Maintain a table containing the filename and the starting address location of that file.
    b. Give options for creating a new file.
    c. Get the name of the file as input from the user. If the file does not already exist, increment the file counter and add the file to the directory.
    d. Update the table accordingly.
3. Tree Structured Directory
    a. Maintain tables for each directory starting from root.
    b. Create a structure for a node in tree which contains an array to hold directories and an array to hold files.
    c. Limit each directory to have a maximum of three sub-directories and files.
    d. For each sub-directory follow the same table structure as described above.
    e. Give options for creating a new directory or a new file.
    f. Get the name and path of the directory or file as input from the user.
    g. Update the table accordingly.
4. Stop

Code:

```
//Program to implement memory organisation techniques
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

#define MAX 100
```

```
#define MAX_DIR 3
#define MAX_FILE 3

typedef struct File
{
   char name[25];
   int start_address;
} File;

void insertFileSingleLevel(File *[]);
void displaySingleLevel(File *[]);

typedef struct Directory
{
   char name[25];
   struct Directory *subdir[MAX_DIR];
   File *f[MAX_FILE];
} Directory;

void init_dir(Directory *const);
void insertFileTree(Directory *const);
void insertDirectoryTree(Directory *const);
void displayTree(const Directory *const, char path[]);

int main()
{
   int choice, count = 0;
   char name[30];
   char path[100];

   File *arr[MAX], *tmp = NULL;
   for (int i = 0; i < MAX; i++)
      arr[i] = NULL;

   Directory root;
   init_dir(&root);
   strcpy(root.name, "root");

   while (1)
   {
      printf("\n\t\t\tFILE ORGANISATION TECHNIQUES\n");
      printf(" 1 - Single Level Directory\n");
      printf(" 2 - Tree Structure Directory\n");
      printf(" 0 - Exit\n");
      printf(" --------------------------------\n");
      printf(" Enter your choice: ");
      scanf("%d", &choice);

      switch (choice)
```

```
{
case 0:
   exit(0);
case 1:
   while (1)
   {
      printf("\n\n\t\tSINGLE LEVEL DIRECTORY\n");
      printf(" 1 - Create a file\n");
      printf(" 2 - List all files\n");
      printf(" 0 - Back\n");
      printf(" ----------------------------\n");
      printf(" Enter your choice: ");
      scanf("%d", &choice);
      getchar();
      if (choice == 0)
         break;

      switch (choice)
      {
      case 1:
         insertFileSingleLevel(arr);
         break;
      case 2:
         displaySingleLevel(arr);
         break;
      default:
         printf(" Invalid Input!\n");
      }
   }
   break;
case 2:
   while (1)
   {
      printf("\n\n\t\tTREE STRUCTURE DIRECTORY\n");
      printf(" 1 - Create a file\n");
      printf(" 2 - Create a directory\n");
      printf(" 3 - List all files\n");
      printf(" 0 - Back\n");
      printf(" ----------------------------\n");
      printf(" Enter your choice: ");
      scanf("%d", &choice);
      getchar();
      if (choice == 0)
         break;

      switch (choice)
      {
      case 1:
         insertFileTree(&root);
```

```
            break;
         case 2:
            insertDirectoryTree(&root);
            break;
         case 3:
            strcpy(path, "/root");
            printf(" +-------------------------+-----------------------------------+\n");
            printf(" |         File Name       |                Path               |\n");
            printf(" +-------------------------+-----------------------------------+\n");
            displayTree(&root, path);
            printf(" +-------------------------+-----------------------------------+\n");
            break;
         default:
            printf(" Invalid Input!\n");
         }
      }
      break;
   default:
      printf(" Invalid Input!\n");
      break;
   }
  }
}

void init_dir(Directory *const dir)
{
  strcpy(dir->name, "");
  for (int i = 0; i < 3; i++)
    dir->f[i] = dir->subdir[i] = NULL;
}

void insertFileSingleLevel(File *root[])
{
  File *tmp = (File *)malloc(sizeof(File));
  printf(" Enter the name of the file: ");
  scanf("%[^\n]", tmp->name);
  tmp->start_address = 500 * (random() % 20);

  int found = 0;

  for (int i = 0; i < MAX; i++)
    if (root[i] == NULL)
    {
      root[i] = tmp;
      break;
    }
    else if (strcmp(root[i]->name, tmp->name) == 0)
    {
      found = 1;
```

```
            break;
        }

    if (found)
        printf(" Duplicate file name!\n");
    else
        printf(" Successfully added file!\n");
}

void displaySingleLevel(File *root[])
{
    if (!root[0])
        printf(" Empty Directory!\n");
    else
    {
        printf(" +--------------------------+---------------+\n");
        printf(" |          File Name       | Start Address |\n");
        printf(" +--------------------------+---------------+\n");
        for (int i = 0; i < MAX && root[i]; i++)
            printf(" | %-25s |     %-4d      |\n", root[i]->name, root[i]->start_address);
        printf(" +--------------------------+---------------+\n");
    }
}

void insertDirectoryTree(Directory *const root)
{
    char path[100];
    printf(" Enter path to directory [root/.../...]: ");
    scanf("%[^\n]", path);

    char *dir, *new_dir;
    Directory *cd = root;

    int found = 0, created = 0;

    dir = strtok(path, "/");
    if (strcmp(path, "root"))
    {
        printf(" Path should start with root!\n");
        return;
    }
    dir = strtok(NULL, "/");
    if (!dir)
    {
        printf(" \nInvalid Directory Name!\n");
        return;
    }
    while (dir != NULL)
    {
```

```
        for (int i = 0; i < MAX_DIR; i++)
        {
            if (cd->subdir[i])
                if (strcmp(dir, cd->subdir[i]->name) == 0)
                {
                    cd = cd->subdir[i];
                    found = 1;
                    break;
                }
        }
        new_dir = dir;
        dir = strtok(NULL, "/");
        if (!found)
            break;
    }
    if (dir == NULL)
    {
        for (int i = 0; i < MAX_DIR; i++)
            if (!cd->subdir[i])
            {
                cd->subdir[i] = (Directory *)malloc(sizeof(Directory));
                init_dir(cd->subdir[i]);
                strcpy(cd->subdir[i]->name, new_dir);
                created = 1;
                break;
            }
            else if (strcmp(cd->subdir[i]->name, new_dir) == 0)
                break;
    }

    if (created)
        printf(" Successfully created directory!\n");
    else
        printf(" Unable to create directory!\n");
}

void insertFileTree(Directory *const root)
{
    char path[100];
    printf(" Enter path to files [root/.../...]: ");
    scanf("%[^\n]", path);

    char *dir, *new_file;
    Directory *cd = root;

    int found = 0, created = 0;

    dir = strtok(path, "/");
    if (strcmp(path, "root"))
```

```
    {
        printf(" Path should start with root!\n");
        return;
    }
    dir = strtok(NULL, "/");
    while (dir != NULL)
    {
        for (int i = 0; i < MAX_DIR; i++)
        {
            if (cd->subdir[i])
                if (strcmp(dir, cd->subdir[i]->name) == 0)
                {
                    cd = cd->subdir[i];
                    found = 1;
                    break;
                }
        }
        new_file = dir;
        dir = strtok(NULL, "/");
        if (!found)
            break;
    }
    if (dir == NULL)
    {
        for (int i = 0; i < MAX_DIR; i++)
            if (!cd->f[i])
            {
                cd->f[i] = (File *)malloc(sizeof(File));
                strcpy(cd->f[i]->name, new_file);
                created = 1;
                break;
            }
            else if (strcmp(cd->f[i]->name, new_file) == 0)
                break;
    }

    if (created)
        printf(" Successfully created File!\n");
    else
        printf(" Unable to create File!\n");
}

void displayTree(const Directory *dir, char path[100])
{
    for (int i = 0; i < MAX_FILE; i++)
        if (dir->f[i])
            printf(" | %-25s | %-35s |\n", dir->f[i]->name, path);

    for (int i = 0; i < MAX_DIR; i++)
```

```
        if (dir->subdir[i])
        {
            strcat(path, "/");
            strcat(path, dir->subdir[i]->name);
            displayTree(dir->subdir[i], path);
        }
}
```

Output:

```
root@hadoop-slave-3:~/krith# ./org

                        FILE ORGANISATION TECHNIQUES
1 - Single Level Directory
2 - Tree Structure Directory
0 - Exit
--------------------------------
Enter your choice: 1


                SINGLE LEVEL DIRECTORY
1 - Create a file
2 - List all files
0 - Back
-----------------------------
Enter your choice: 1
Enter the name of the file: newfile1
Successfully added file!


                SINGLE LEVEL DIRECTORY
1 - Create a file
2 - List all files
0 - Back
-----------------------------
Enter your choice: 1
Enter the name of the file: newfile1
Duplicate file name!


                SINGLE LEVEL DIRECTORY
1 - Create a file
2 - List all files
0 - Back
-----------------------------
Enter your choice: 1
Enter the name of the file: newfile2
Successfully added file!

                SINGLE LEVEL DIRECTORY
1 - Create a file
2 - List all files
0 - Back
-----------------------------
Enter your choice: 2
+---------------------------+---------------+
|          File Name        | Start Address |
+---------------------------+---------------+
| newfile1                  |     1500      |
| newfile2                  |     8500      |
+---------------------------+---------------+


                SINGLE LEVEL DIRECTORY
1 - Create a file
2 - List all files
0 - Back
-----------------------------
Enter your choice: 0
```

```
               TREE STRUCTURE DIRECTORY
1 - Create a file
2 - Create a directory
3 - List all files
0 - Back
-----------------------------
Enter your choice: 1
Enter path to files [root/.../...]: root/home/file1
Unable to create File!


               TREE STRUCTURE DIRECTORY
1 - Create a file
2 - Create a directory
3 - List all files
0 - Back
-----------------------------
Enter your choice: 2
Enter path to directory [root/.../...]: root/home
Successfully created directory!


               TREE STRUCTURE DIRECTORY
1 - Create a file
2 - Create a directory
3 - List all files
0 - Back
-----------------------------
Enter your choice: 1
Enter path to files [root/.../...]: root/home/file1
Successfully created File!
```

```
               TREE STRUCTURE DIRECTORY
1 - Create a file
2 - Create a directory
3 - List all files
0 - Back
-----------------------------
Enter your choice: 3
+---------------------------+-------------------------------------+
|          File Name        |                 Path                |
+---------------------------+-------------------------------------+
| file1                     | /root/home                          |
+---------------------------+-------------------------------------+


               TREE STRUCTURE DIRECTORY
1 - Create a file
2 - Create a directory
3 - List all files
0 - Back
-----------------------------
Enter your choice: 1
Enter path to files [root/.../...]: root/home/file2
Successfully created File!


               TREE STRUCTURE DIRECTORY
1 - Create a file
2 - Create a directory
3 - List all files
0 - Back
-----------------------------
Enter your choice: 3
+---------------------------+-------------------------------------+
|          File Name        |                 Path                |
+---------------------------+-------------------------------------+
| file1                     | /root/home                          |
| file2                     | /root/home                          |
+---------------------------+-------------------------------------+
```

```
                TREE STRUCTURE DIRECTORY
1 - Create a file
2 - Create a directory
3 - List all files
0 - Back
-----------------------------
Enter your choice: 2
Enter path to directory [root/.../...]: root/user
Successfully created directory!


                TREE STRUCTURE DIRECTORY
1 - Create a file
2 - Create a directory
3 - List all files
0 - Back
-----------------------------
Enter your choice: 1
Enter path to files [root/.../...]: root/user/file1
Successfully created File!


                TREE STRUCTURE DIRECTORY
1 - Create a file
2 - Create a directory
3 - List all files
0 - Back
-----------------------------
Enter your choice: 3
+---------------------------+-------------------------------------+
|          File Name        |                 Path                |
+---------------------------+-------------------------------------+
| file1                     | /root/home                          |
| file2                     | /root/home                          |
| file1                     | /root/home/user                     |
+---------------------------+-------------------------------------+


                TREE STRUCTURE DIRECTORY
1 - Create a file
2 - Create a directory
3 - List all files
0 - Back
-----------------------------
Enter your choice: 0

                      FILE ORGANISATION TECHNIQUES
1 - Single Level Directory
2 - Tree Structure Directory
0 - Exit
---------------------------------
Enter your choice: 0
root@hadoop-slave-3:~/krith#
```

Learning outcomes:

- File organisation techniques were understood and implemented.
- Single level and hierarchical level organisation was understood and implemented.