Maharashtra Academy of Engineering & Educational Research's

MIT COLLEGE OF ENGINEERING, PUNE-411 038.

Name: Gauri Karekar Class : IE IT Batch :

Subject SL-2 Roll No. : 3154169 Exp. No : 1

Name of the Experiment

Performed on

Submitted on :

Marks	Teacher's Signature with date
9	
10	

Incomplete for Theory ☐ Diagrams ☐ Observation Table ☐
Calculations ☐ Graphs ☐ Results ☐ Conclusion ☐
Understanding Through Q/A ☐ Late Submission ☐ Neatness ☐

Teacher's Signature

Theory :-

Shell

In computing, a shell is a user interface for access to an operating system's services. In general, operating system shells use either a command-line interface (CLI) or graphical user interface (GUI), depending on a computer's role and particular operation. It is named as a shell because it is a layer around the operating system kernel.

Shell scripts :

A shell script is a computer program designed to be be run by the Unix shell, a command-line interpreter. The various dialects of shell scripts are considered to be scripting languages.

A script is defined as just a plain text file or ASCII file - with a set of linux/unix commands
- flow of control
- I/O facilities

A shell script can be created using any text editor.
Shell script allows use of variables.
Shell scripts are interpreted directly and are not compiled as C/C++ codes.
Shells provide many features including loop constructs, arrays, variables, branches and functions.

Advantages

- Shell script is much quicker than programming in any other languages.
- To automate the frequently performed tasks.
- Easy to understand and use.

Disadvantages

- Slow execution speed.
- Prone to costly errors.
- Compatibility problems.

To successfully write a shell script, you have to do three things :-

1. Write a script.
2. Give the shell permission to execute it.
3. Put it somewhere the shell can find.

The following is a list of some shell commands:

1) acroread - Read or print a PDF file.
2) cat - Send a file to the screen in one go. Useful for piping to other programs.
   eg: cat file1 - list file1 to screen
   cat file1 file2 > outfile - add files together into outfile
3) cc - Compile a C program.
4) cd - change current directory
5) cp - copy file(s)
   eg: cp file1 file2 - copies file1 to file2
6) date - Shows current date
7) dvips - Convert a dvi file to PostScript
8) emacs - The ubiquitous text editor
9) file - Tells you what sort of file it is.
10) grep - looks for text in files. List out lines containing text
11) kill - kill, pause or continue a process. Can also be used for killing daemons.
12) lp - sends files to a printer
13) mv - move or rename files.

- Compile Command

To execute a .sh file we need to grant permissions.

$ chmod 755 filename.sh

(755 grants read write & execute permission to .sh file.)

$ ./filename.sh to execute

Maharashtra Academy of Engineering & Educational Research's

MIT COLLEGE OF ENGINEERING, PUNE-411 038.

Name: Gauri Karekar Class : TE IT Batch :

Subject SL-2 Roll No. : 3154169 Exp. No : 2

Name of the Experiment

Performed on

Submitted on :

Marks	Teacher's Signature with date
8	
10	

Theory :-

Process Control System Call

A process is a unit of work.

Process control involves creating new processes to work concurrently on solving different problems.

	Windows	Unix
Process Control	CreateProcess()	fork()
	ExitProcess()	exit()
	WaitForSingleObject()	wait()

In Unix, there are four main system calls for process control - fork, exit, execve, wait

fork : create new process

exit : normal termination of process (exit is not actually a system call, but it calls _exit, which is)

execve : load and execute another program

wait : wait for another process to finish

fork()

System call fork() is used to create processes.

It takes no arguments and returns process ID.

The purpose of fork() is to create a new process, which becomes the child process of the caller.

After a new child process is created, both processes will execute the next instruction following the fork() system call.

Therefore we have to distinguish the parent from the child.

This can be done by testing the returned value of fork().

If fork() returns a -ve value, creation of child process was unsuccessful.

fork() returns a zero to the newly created child process.

fork() returns a +ve value, the process ID of the child process to the parent. The returned process ID is of the type pid_t defined in sys/types.h

Normally the process ID is an integer.

execve()

execve() executes the program pointed to by filename. filename must be either a binary executive or executable or a script starting with a line of the form "#! interpreter [arg]".

execve - execute program

Syntax :-

```
#include <unistd.h>
int execve(const char *filename, char *const argv[],
           char *const envp[]);
```

argv is an array of argument strings passed to the new program.

envp is an array of strings, conventionally of the form key=value, which are passed as environment to the new program

wait()

The function blocks the calling process until one of its child processes exits or a signal is received.

wait() takes the address of an integer variable and returns the process ID of the completed process.

One of the main purposes of wait() is to wait for child process.

- Zombie process

a) A zombie process or defunct process is a process that has completed execution but still has an entry in the process table

b) It is a process in the terminated state

c) This occurs for child processes, where the entry is still needed to allow the parent process to read it's child's exit status.

d) Once the exit status is read via the wait system call, the zombie's entry is removed from the process table and it is said to be "reaped".

e) The term zombie derives from the common definition of zombie - an undead person.

- Orphan process :

An orphan process is a computer process whose parent process has finished or terminated, though it remains running itself.

- Creation of child process :-

i) A process can create new child process using fork() system call

ii) This child process created through fork() call will have same memory image as of parent process i.e. it will be duplicate of calling.

iii) As memory image of new child process will be copy of parent process's ~~memo~~ memory image.
iv) So all variables defined before fork() call will be available in child process with such values.
v) If fork() call is successful then call after this call will be executed in both the process.
vi) Thus fork() function's return value will be executed in both the process.
vii) Thus fork() function's return value will be different in both the process, i.e.
a) if fork() call is successful, it will return 0 in child process and return process id of new child process in parent process.
b) if fork() call is unsuccessful then it will be -1

Example

```
#include<unistd.h>
int main()
{
 std :: cout <<"current process Id : ";
 pid_t childprocessId =fork();
 if (childprocessId < 0)
  {
    std:: cout<< "failed to create a new process ";
  }
```

```
else if (childprocessId == 0)
{
    std::cout << "Child Process";
}

else if (childprocessId > 0)
{
    std::cout << "Parent Process";
}
}
```

ame: Gauri Karekar Class : TE IT Batch :

ubject SL-2 Roll No. 8154169 Exp. No : 3

ame of the Experiment

erformed on 24/7/17

ubmitted on : 24/8/17

Marks	Teacher's Signature with date
8	24/8/17.
10	

* Implement multithreading for matrix multiplication using Pthreads

* Theory :-

- Threads : i) A process is a program that performs a single thread of execution.
  ii) eg: when a process is running a word-process program, a single thread of instruction is being processed.
  iii) This single thread of control allows the process to perform only one task at a time.
  iv) All threads of a process share global variables, file descriptions, signal handler and its current directory state with its creator.
  v) Suspending a process involves suspending all threads of that process. Since all threads share the same address space
  vi) Termination of a process terminates all the threads.

- P-thread library - Pthreads are defined as a set of C-language programming types and procedure calls, implemented with a pthread.h header / include file and a thread library - through this library, such as libc, in some implementations.

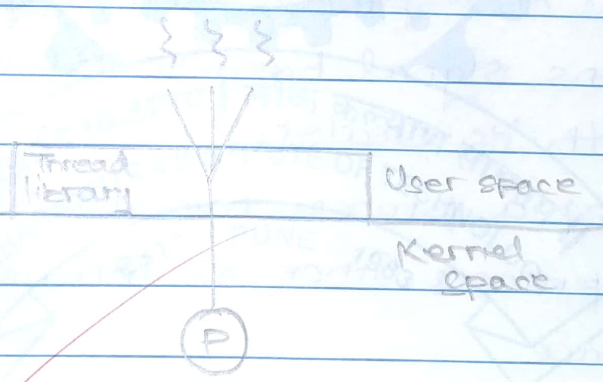
- Types of threads are User level and Kernel level threads.

i) User level threads

a) All thread management is done by the application

b) The kernel is not aware of the existence of threads.

c) Thread library contains code for :-
- creating & destroying threads
- message and data passing
- scheduling ~~or~~ thread execution
- saving and restoring thread context.

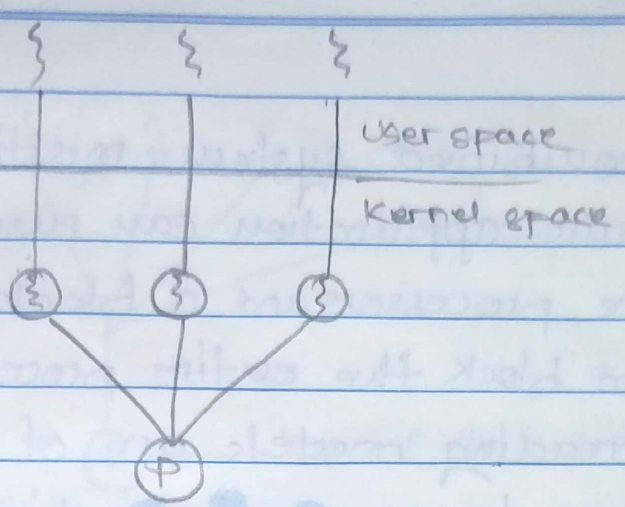


ii) Kernel level threads:

a) The Kernel maintains context information for the process & the threads

b) example: ~~or~~ Windows

c) Scheduling is done on a thread basis



iv) To make concurrency cheaper, the execution aspect of processing is seperated out into threads.

v) As such the OS now manages threads and processes.

vi) All the thread operations are implemented in the kernel and the OS schedules all threads in the system.

vii) OS managed threads are called kernel-level threads or light weight processes.

- Multithreading

i) Multithreaded programming provides a mechanism for more efficient use of multiple cores and improved concurrency.

ii) Multithreading is the ability of a program or an OS process to manage its use by more than one user at a time and to even manage multiple requests by the same user without having to have multiple ~~copie~~ copies of the programming running in the computer.

iii) In a combined system, multiple threads within the same application can run in parallel on multiple processors and a blocking system call need not block the entire process.

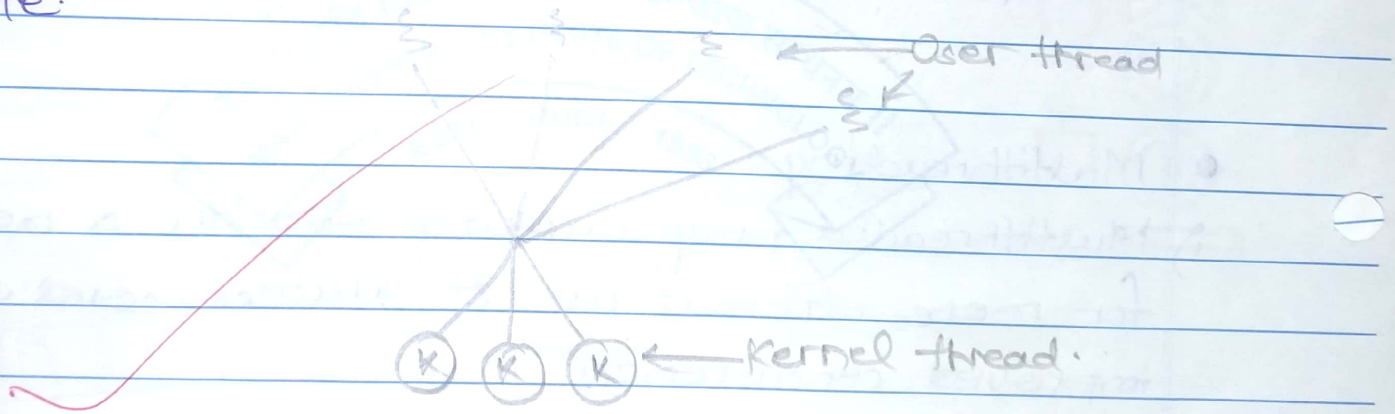
iv) Multithreading models are of 3 types -

a) Many to many relationship
b) Many to one relationship
c) One to one relationship

a) Many-to-Many model

i) This model multiplexes many user level threads to a smaller or equal number of kernel threads.

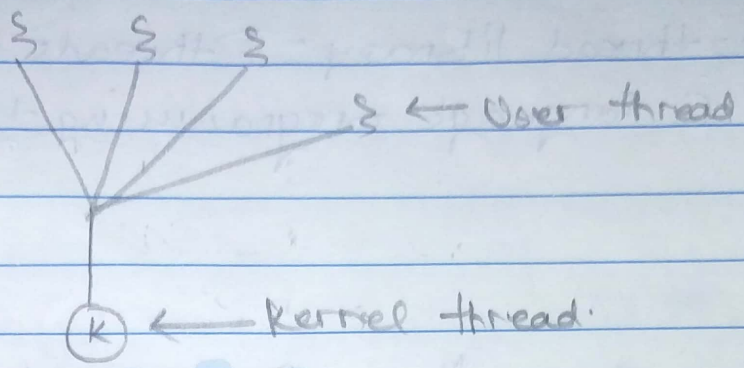
ii) The number of kernel threads may be specific to either a particular application or a particular file.



b) Many-to-one model

i) It maps many user level threads to one kernel thread.

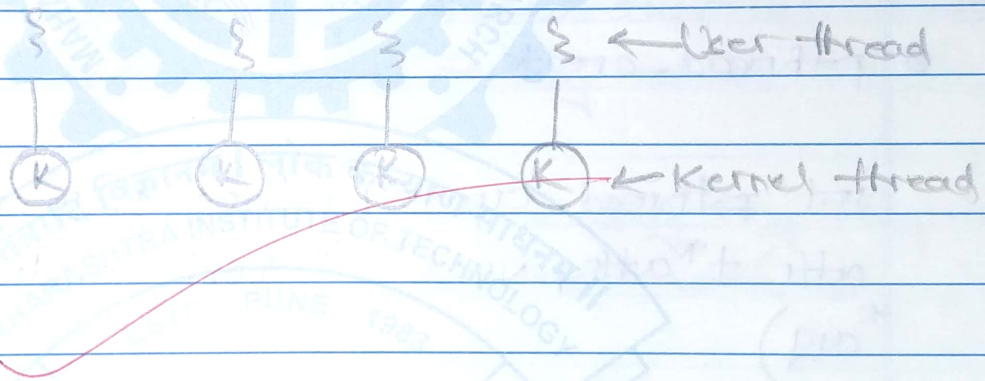
ii) Thread management is done by thread library in user space.



c) One - to - one model

i) This model maps each user thread to a kernel thread.

ii) It provides more concurrency than the many to one model.



- POSIX

It is an acronym of Portable Operating System Interface. It is a family of standards specified by IEEE computer society for maintaining compatibility between OS. POSIX defines the application programming interface (API), along with command line shells to utility interfaces, software compatibility with variats of UNIX and other OS.

Depending upon the degree of compliance with the standards, one can classify OS as fully or partly POSIX compatible.

P-thread library - Pthreads are defined as a set of C-language programming types

They share the process address space - no initialization for a new system virtual memory space.
They also gain performances on uniprocessor since one thread may execute while another is waiting for I/O.

pthread_create

int pthread_create (pthread_t* thread, const pthread_attr_t* attr, void* (*start_routine)(void*), void *arg)

pthread create arguments:
thread: a unique identifier for the new thread returned by the subroutine
attr: an attribute object that may be used to set thread attributes.
start_routine: the C routine that the thread will execute once it created.
arg: a single argument that may be passed to start_routine. It must be passed by reference as a pointer cast of type void.

- on success, pthread_create returns 0; on error it returns an error number.

- pthread_join :-

i) int pthread_join (pthread_t thread, void ** retval);

ii) Arguments
thread: A unique identifier for the new thread returned by the subroutine.

retval :- If retval is not null, then pthread_join ( ) copies the exit status of the target thread supplied to pthread_exit into the location pointed to by retval.

iii) If multiple threads simultaneously try to join with the same thread, the results are undefined.

iv) On success, pthread_join ( ) returns 0; on error it returns an error number.

- pthread_exit( );

i) The pthread_exit( ) routine allows the programmer to specify an optional termination status parameter.
ii) This optional parameter is typically returned to threads "joining" the terminated thread.
iii) calling pthread_exit( ) from main ( ):
a) There is a definite problem if main( ) finishes

before the threads it spawned if you don't call pthread_exit() explicitly.

5) By having main() explicitly call pthread_exit() as the last thing it does main() will blur and be kept alive to support the threads if created until they are done.

- Compilation Command:

i) gcc program.c -o pname -

ii) By default gcc does not include the pthread library. So you have to include the library using pthread argument

conclusion

Conclusion:- Multithreading for matrix multiplication has been implemented using pthreads.

Maharashtra Academy of Engineering & Educational Research's

MIT COLLEGE OF ENGINEERING, PUNE-411 038.

Name: Gauri Karekar Class : TE IT Batch :

Subject SL - 2 Roll No. : 3154169 Exp. No : 4

Name of the Experiment

Performed on 24/8/17

Submitted on : 28/8/17

Marks	Teacher's Signature with date
9	28/8/17
10	

Thread synchronization using counting semaphores and mutual exclusion using mutex.

Theory :-

Synchronization refers to one of the two distinct but related concepts: synchronization of processes, and synchronization of data. Process synchronization refers to the idea that multiple processes are to join up at a certain point in order to reach an agreement or commit to a certain sequence of action. Data synchronization refers to the idea of keeping multiple copies of a dataset in coherence with one another, or to maintain data integrity.

Synchronization means sharing system resources by processes in such a way that concurrent access to shared data is handled nearby minimizing the chance of inconsistent data. Maintaining data consistency demands mechanisms to ensure.

Semaphores :- This is a technique for managing concurrent processing by using the value of a simple integer variable.

Its basically a synchronization tool and is accessed only through two low standard atomic operations, wait and signal designated by P() P() and V() respectively.

Wait() - decrement the value of its argument, as soon as it would become non-negative.

Signal() - increment the value of its argument as an individual operation.

Semaphores are mainly of two types:-

Binary semaphore: It is a special form of semaphore used for implementing mutual exclusion hence its often called as mutex. A binary semaphore is initialized to 1 and only takes value 0 and 1 during execution of a process.

Counting semaphore: These are used to implement bounded concurrency.

All POSIX semaphore functions and types are prototyped or designed in <semaphore.h>

sem_init() - It is used to initialize a semaphore.

int sem_init (sem_t *sem, int pshared, usigned int value);

sem - points to a semaphore object to initialize
pshared - is a flag indicating whether or not the semaphore should share itself with fork() process.
value - is an initial value to set the semaphore.

sem_wait() - it is used to wait a semaphore
int sem_wait (sem_t *sem);

sem_post() - to increment the value of semaphore
int sem_post (sem_t *sem);
wakes the blocked process.

destroy() - to destroy a semaphore
int sem_destroy (sem_t *sem)
no threads should be waiting on the semaphore if its destruction is to succeed.
sem_t is a datatype in <semaphore.h> to initializ
a semaphore variable.

pthread_mutex_destroy() - destroys the mutex object referenced by mutex.

int pthread_destroy(pthread_mutex_t *mutex);

Conclusion :- Thread synchronization has been implemented using counting semaphores and mutual exclusion using mutex.

Maharashtra Academy of Engineering & Educational Research's

MIT COLLEGE OF ENGINEERING, PUNE-411 038.

Name: Gauri Karekar Class : TE IT Batch :

Subject SL-2 Roll No. : 3154169 Exp. No : 5

Name of the Experiment

Performed on 31/8/17

Submitted on : 8/9/17

Marks	Teacher's Signature with date
10	8/9/17
10	

Theory :-

File structure : A file structure should be according to a required format that the operating system can understand.

A file is a sequence of characters organized into lines.

A source file is a sequence of producers and functions

An object file is a sequence of bytes organized into blocks that are understandable by machine.

fopen() :- a library function that opens the filename pointed to it using given mode.

*fp = FILE *fopen (const char *filename, const char *mode);

fclose() :- This is used to close an open file.

feof() :- It tests the end of file for the given stream.

fgetc() - it gets the next character (an unsigned char) from the specified stream and advances the position indicator for the stream

putc() - The function writes a character specified by argument char to the specified stream and advances the pointer indicator for the stream

Reader's Writer's Problem - Pseudocode

Writer Process :-

```
wait (wrt);
    ... writing is performed
    ...
signal (wrt);
```

Readers Process :

```
wait (mutex)
    read count ++ ;
    if readcount =1 then wait (wrt)
    signal (mutex)
    ... reading is performed ...
```

```
wait (mutex)
    readcount --;
if (readcount == 0) then signal (wrt);
signal (mutex);
```

Conclusion :- Readers Writers problem studied and implemented successfully.

Maharashtra Academy of Engineering & Educational Research's

MIT COLLEGE OF ENGINEERING, PUNE-411 038.

Name: Gauri Karekar Class : TE IT Batch :

Subject SL-2 Roll No. : 3154169 Exp. No : 6

Name of the Experiment

Performed on 8/9/17

Submitted on : 14/9/17

Marks	Teacher's Signature with date
9	14/9/17
10	

Incomplete for Theory ☐ Diagrams ☐ Observation Table ☐

Graphs ☐ Results ☐ Conclusion ☐

Theory :-

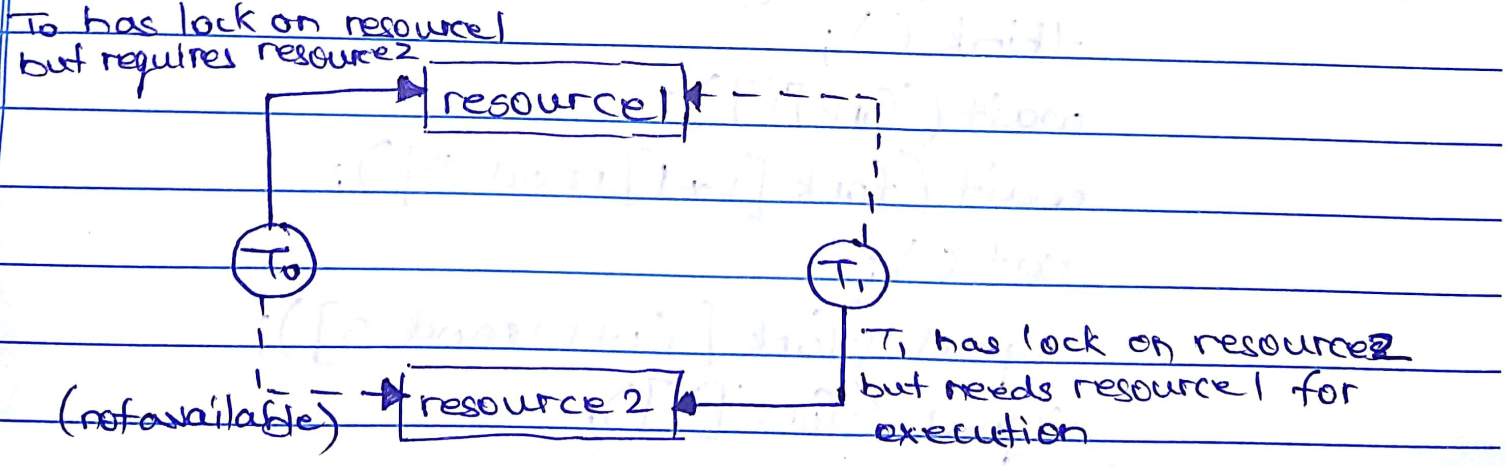
Deadlock :-

- Deadlocks are set of blocked processes each holding a resource and waiting to acquiring a resource held by another process.
- Deadlocks can be avoided by atleast one of the following conditions, because all these four conditions, ~~because~~ are required simultaneously to cause deadlock :-

1) Mutual Exclusion
2) Hold & wait
3) No pre-emption
4) Circular wait

These are the three strategies to remove deadlock after its occurrence

1) Preemption 2) Rollback 3) Kill one or more process



Starvation :-

Also referred to as indefinite blocking is phenomenon associated with the priority scheduling algorithms, in which a process ready to run for CPU can wait indefinitely because of low priority.

Dining philosopher problem :-

Its a classic synchronization problem used to evaluate situations where there is need of allocating multiple resources to multiple processes.

```
Semaphore fork[5] = {1};
int i;
void philosopher(int i).
{
    while (true) {
        think();
        wait(fork[i]);
        wait(fork[i+1] mod 5]);
        eat();
        signal(fork[(i+1) mod 5]);
        signal(fork[i]);
    }
```

```
}

void main()
{
  parbegin (philosopher(0); philosopher(1);
  philosopher(2); philosopher(3); philosopher(4));
}
```

Conclusion:- Dining Philosopher's problem studied and implemented successfully.

Maharashtra Academy of Engineering & Educational Research's

MIT COLLEGE OF ENGINEERING, PUNE-411 038.

Name: Gauri Karekar Class : TE IT Batch :

Subject SL - 2 Roll No. : 8154169 Exp. No : 7

Name of the Experiment

Performed on

Submitted on :

Marks	Teacher's Signature with date
10	
10	

Incomplete for Theory ☐ Diagrams ☐ Observation Table ☐

Calculations ☐ Graphs ☐ Results ☐ Conclusion ☐

Understanding Through Q/A ☐ Late Submission ☐ Neatness ☐

Teacher's Signature

Title - Interprocess Communication

Theory :-

- IPC :- Interprocess Communication (IPC) is a set of programming interfaces that allow a programmer to coordinate activities among different program processes that can run concurrently in operating system.

a) Pipes :-

- Pipe :- Pipe is a simple FIFO communication channel that may be used for one-way IPC. An implementation is often integrated into the OS's file I/O subsystem.

- System call :-

```
int pipe (int fd[2]);
```

fd[0] will be the file descriptor for the read end of pipe.

fd[1] will be the file descriptor for the write end of pipe.

Returns 0 on success -1 on error.

open ()

#include <fcntl.h>
int open (char *filename, int access, int permissions)

modes are
O_RDONLY O_WRONLY O_RDWR O_APPEND

permission are
S_IWRITE

read()
#include <fcntl.h>
int read (int handle, buffer, int nbytes)

handle is file descriptor, buffer is a memory space to store contents and nbyte is no. of bytes to be read.

from buffer to the file associated with handle on text file, it expands each LF to CR/LF

close ()

int close (int handle);

The close (); function closes the file associated with handle. The function returns 0 if successful -1 to indicate error.

FIFO :- Named pipe also known as FIFO for its behaviour is an extension to the traditional pipe concept on UNIX and UNIX-like system, and is one of the methods of user-process communication (IPC)

shell command :- $ mkfifo and mknod are used to create a named pipe

mkfifo :- mkfifo create fifo - the named pipes

Syntax :- mkfifo [options] fifo_name
eg:- mkfifo fifo

- mknod :- used to create a named pipe.

$ mknod [option] ... name type
$ mknod fifo1 p

- unlink ( ) : It deletes a name from the file system

If the name referred to a socket, fifo or device the name for it is removed but processes which have the object open may continue to use it

On success 0 is returned -1 on error and error no is set appropriately.

- Unnamed pipe

1) These are created by the shell automatically.
2) They exist in the kernel.
3) They can not be accessed by any process.
4) They are unidirectional

- Named pipe

1) They are created programatically using command mkfifo
2) They exist in the file system with a given file name
3) They can be viewed and accessed by any two un-related processes

They are not opened while creation.
They are bidirectional.

File pointer
It is high level interface
Passed to fread() and furite() functions
Includes buffering, error detection and EOF detection
Provides higher portability & efficiency.

File descriptor :-
Low/ Kernel level handles
Passed to read() and write() of UNIX system calls.
Doesn't include buffering and such features.
Less portable and lacks efficiency

Conclusion:- Studied and implemented named and unnamed pipes.

Maharashtra Academy of Engineering & Educational Research's

MIT COLLEGE OF ENGINEERING, PUNE-411 038.

Name: Gauri Karekar Class : TE IT Batch :

Subject SL-2 Roll No. : 3154169 Exp. No : 8

Name of the Experiment

Performed on 14/9/17

Submitted on :

Marks	Teacher's Signature with date
10	18/9/17
10	

Theory:-

System V

It is one of the most commercial versions of the Unix Operating System. It was originally developed by AT&T.

Four major versions were released, numbered 1, 2, 3, 4. System V Release 4 was commercially the most successful version being the result of 'Unix System Unification'.

key_t :

Unix requires a key of type key_t defined in file sys/types.h for requesting resources such as shared memory segments, message queues and semaphores. A key is simply an integer of type key_t.

The length of a key is system independent.

In order to create a new message queue, or access an existing queue, the shmget() system call is used.

shmget() - allocates a System V shared memory segment

synopsis :-

```
#include <sys/ipc.h>
#include <sys/shm.h>

int shmget(key_t key, size_t size, int
                                shmflg);
```

shmat(), shmdt()

They are shared memory operations

synopsis:-

```
#include <sys/types.h>
#include <sys/shm.h>
  void *shmat (int shmid, const void* shmaddr,
     int shmflg);
  int shmdt (const void *shmaddr);
```

shmat()

It attaches the shared memory segment identified by shmid to the address space of the calling process.
The attaching address is specified by shmaddr

shmdt()

Conclusion:- Interprocess communication in Linux using shared memory using System V successfully implemented.

Maharashtra Academy of Engineering & Educational Research's

MIT COLLEGE OF ENGINEERING, PUNE-411 038.

Name: Gauri Karekar Class : TE IT Batch :

Subject SL-2 Roll No. : 3154169 Exp. No : 9

Name of the Experiment

Performed on

Submitted on : 25/9/17

Marks	Teacher's Signature with date
10	
10	25/9/2017.

Incomplete for Theory ☐ Diagrams ☐ Observation Table ☐
Calculations ☐ Graphs ☐ Results ☐ Conclusion ☐

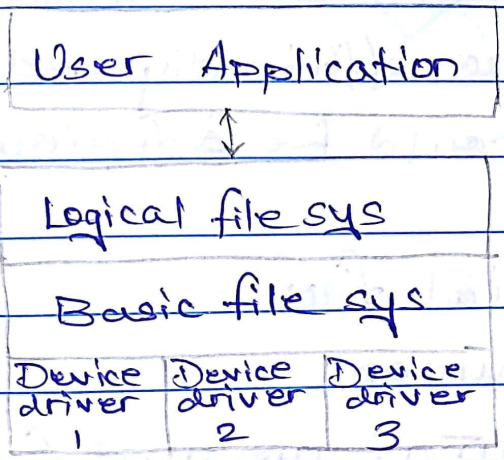
Implement a new system call in the kernel space, add this new system call in the Linux kernel by the compilation of this kernel (any kernel source, any architecture, and any Linux kernel distribution) and demonstrate the use of this embedded system call using C program in user space:

Theory :-

File System Architecture

The File System Architecture specifies how the files will be stored into the computer system

Most file systems use hierarchical organization naturally.



- Device drivers : communicate directly with peripheral devices; responsible for starting physical

I/O operations on devices; processes the completion of an I/O request.

- Basic file system :-
  - uses specific device driver
  - deals with blocks of data that are exchanged with physical device.

- Logical file system :- Responsible for providing interface to the user including
  a) file access b) directory operations c) Security & protection

File Organization Types

I] The pile
- data are collected in the same order they arrived
- accumulates mass of data and saves it
- uses space efficiently and is updated easily
- retrival could be tedious.

II] Sequential file
- most common
- fixed format are used for records and records are of the same length
- simplest structure, easy to implement. Accessing a single record takes a long time.

III] Indexed Sequential file
- allows records to be accessed either sequentially or randomly (with an index)
- a secondary set of hash tables known as indexes is created that contains pointers to the main file.
- each file has an index to support random search.

IV] Direct access file
- accesses directly any file block of a known address.
- key field is required for each record.
- uses hashing on the key value.

- open()
- open and possibly create a file or device
- given a pathname for a file, open() returns a file descriptor, a small, non-negative integers for

- read()
- read from file descriptor
- attempts to read up to count bytes from file descriptor fd into buffer.

- write()
- writes to a file descriptor

- close()
  - closes a fd
  - closes a fd, so that it no longer refers to any file and maybe reused. Any record locks held on the file it was associated with and curved by process are removed.

- lseek()
  - reposition read/write file offset.
  - attempts to reposition the offset of the open file associated with fd

Conclusion :- Studied about file organization types and system calls.

Maharashtra Academy of Engineering & Educational Research's

MIT COLLEGE OF ENGINEERING, PUNE-411 038.

Name: Gauri Karekar Class : TE IT Batch :

Subject SL-2 Roll No. : 31 Exp. No : 10

Name of the Experiment

Performed on

Submitted on :

Marks	Teacher's Signature with date
10	

Implement a new system call, add this new system call in the Linux kernel (any kernel source and any Linux kernel distribution) and demonstrate the use of same.

Theory :-

System call

A system call is the programmatic way in which a computer program requests a service from the kernel of the operating system is executed on. This may include hardware related services (eg: accessing a hard disk drive), creation and execution of new processes, and communication with integral kernel services such as process scheduling. System calls can provide an essential interface between a process and the operating system.

System call interface

Most programming languages provides a system call interface

It serves as a link to sys calls made available by the OS.

- It intercepts function calls in the API and invokes the necessary system call within the OS.
- Most of the details of the OS interfaces are hidden from the programmer by the API.

System Call table

- It can be thought of as an API for the interface between user space and kernel space.
- The source file where each system call is located is linked to in the column labelled "source".

uname

- get name and information about current kernel

uname -r

- prints the current kernel release