**2) UNIX OS calls**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <sys/stat.h>

#include <dirent.h>

#include <fcntl.h>

int main() {

pid\_t pid;

struct stat statbuf;

DIR \*dir;

struct dirent \*entry;

// Get and print the process ID using getpid()

printf("Process ID: %d\n", getpid());

// Demonstrate opendir and readdir (Listing files in current directory)

printf("Files in the current directory:\n");

dir = opendir(".");

if (dir == NULL) {

perror("opendir");

exit(EXIT\_FAILURE);

}

while ((entry = readdir(dir)) != NULL) {

printf("%s\n", entry->d\_name);

}

closedir(dir);

// Check if a file exists using stat()

if (stat("example.txt", &statbuf) == 0) {

printf("File 'example.txt' exists.\n");

} else {

printf("File 'example.txt' does not exist.\n");

}

// Fork to create a child process

pid = fork();

if (pid == -1) {

// Fork failed

perror("fork");

exit(EXIT\_FAILURE);

}

if (pid == 0) {

// In child process

printf("Child process ID: %d\n", getpid());

// Use exec() to run a new program (simple ls command)

execlp("ls", "ls", "-l", NULL);

// If exec fails

perror("execlp failed");

exit(EXIT\_FAILURE);

} else {

// In parent process

printf("Parent process, waiting for child to finish...\n");

wait(NULL); // Wait for the child process to terminate

printf("Child process has finished.\n");

}

// Open and close a file (example.txt)

int fd = open("example.txt", O\_RDONLY);

if (fd == -1) {

perror("open");

} else {

close(fd);

printf("File 'example.txt' closed.\n");

}

// Exit the program

exit(EXIT\_SUCCESS);

}

**Output:**

Process ID: 12345

Files in the current directory:

.

..

simple\_program.c

example.txt

a\_file.txt

File 'example.txt' exists.

Child process ID: 12346

Parent process, waiting for child to finish...

total 12

drwxrwxrwx 2 user user 4096 Mar 14 12:34 .

drwxrwxrwx 2 user user 4096 Mar 14 12:34 ..

-rw-r--r-- 1 user user 0 Mar 14 12:34 example.txt

-rw-r--r-- 1 user user 0 Mar

3).UNIX commands

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <dirent.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <unistd.h>

#include <fcntl.h>

void list\_files() {

DIR \*dir;

struct dirent \*entry;

// Open current directory

dir = opendir(".");

if (dir == NULL) {

perror("opendir");

exit(EXIT\_FAILURE);

}

// Read and print each entry in the directory

while ((entry = readdir(dir)) != NULL) {

printf("%s\n", entry->d\_name);

}

closedir(dir);

}

void copy\_file(const char \*source, const char \*destination) {

int src\_fd, dest\_fd;

char buffer[1024];

ssize\_t bytes\_read, bytes\_written;

// Open the source file

src\_fd = open(source, O\_RDONLY);

if (src\_fd == -1) {

perror("open source file");

exit(EXIT\_FAILURE);

}

// Open (or create) the destination file

dest\_fd = open(destination, O\_WRONLY | O\_CREAT | O\_TRUNC, S\_IRUSR | S\_IWUSR);

if (dest\_fd == -1) {

perror("open destination file");

exit(EXIT\_FAILURE);

}

// Copy content from source to destination

while ((bytes\_read = read(src\_fd, buffer, sizeof(buffer))) > 0) {

bytes\_written = write(dest\_fd, buffer, bytes\_read);

if (bytes\_written != bytes\_read) {

perror("write error");

exit(EXIT\_FAILURE);

}

}

if (bytes\_read == -1) {

perror("read error");

exit(EXIT\_FAILURE);

}

// Close the files

close(src\_fd);

close(dest\_fd);

printf("File copied from %s to %s\n", source, destination);

}

void grep\_file(const char \*filename, const char \*search\_string) {

FILE \*file;

char line[1024];

// Open the file for reading

file = fopen(filename, "r");

if (file == NULL) {

perror("fopen");

exit(EXIT\_FAILURE);

}

// Read the file line by line and search for the string

while (fgets(line, sizeof(line), file) != NULL) {

if (strstr(line, search\_string) != NULL) {

printf("%s", line);

}

}

fclose(file);

}

int main() {

int choice;

char source[256], destination[256], filename[256], search\_string[256];

// Simple menu for options

printf("Choose a command to run:\n");

printf("1. List files (ls)\n");

printf("2. Copy file (cp)\n");

printf("3. Search string in file (grep)\n");

printf("Enter your choice (1/2/3): ");

scanf("%d", &choice);

getchar(); // Consume the newline left by scanf

switch (choice) {

case 1:

// List files in the current directory

printf("Listing files in the current directory:\n");

list\_files();

break;

case 2:

// Copy a file

printf("Enter source file name: ");

fgets(source, sizeof(source), stdin);

source[strcspn(source, "\n")] = 0; // Remove newline character

printf("Enter destination file name: ");

fgets(destination, sizeof(destination), stdin);

destination[strcspn(destination, "\n")] = 0; // Remove newline character

copy\_file(source, destination);

break;

case 3:

// Search a string in a file

printf("Enter filename to search in: ");

fgets(filename, sizeof(filename), stdin);

filename[strcspn(filename, "\n")] = 0; // Remove newline character

printf("Enter string to search: ");

fgets(search\_string, sizeof(search\_string), stdin);

search\_string[strcspn(search\_string, "\n")] = 0; // Remove newline character

grep\_file(filename, search\_string);

break;

default:

printf("Invalid choice!\n");

break;

}

return 0;

}

**OUTPUT:**

Choose a command to run:

1. List files (ls)

2. Copy file (cp)

3. Search string in file (grep)

Enter your choice (1/2/3): 1

Listing files in the current directory:

.

..

file1.txt

file2.txt

example.txt

Choose a command to run:

1. List files (ls)

2. Copy file (cp)

3. Search string in file (grep)

Enter your choice (1/2/3): 2

Enter source file name: example.txt

Enter destination file name: example\_copy.txt

File copied from example.txt to example\_copy.txt

Choose a command to run:

1. List files (ls)

2. Copy file (cp)

3. Search string in file (grep)

Enter your choice (1/2/3): 3

Enter filename to search in: example.txt

Enter string to search: hello

hello world

this is a hello test

**11) Dead lock avoidance and preventions using bankers algorithm**

#include <stdio.h>

#include <stdbool.h>

// Number of processes and resources

#define P 5 // Number of processes

#define R 3 // Number of resources

// Function to check if the system is in a safe state (Banker's Algorithm)

bool is\_safe\_state(int available[], int maximum[][R], int allocation[][R], int need[][R]) {

int work[R];

bool finish[P] = {false};

int safe\_sequence[P];

int count = 0;

// Initialize work array to available resources

for (int i = 0; i < R; i++) {

work[i] = available[i];

}

// Try to find a process that can finish

while (count < P) {

bool found = false;

for (int p = 0; p < P; p++) {

if (!finish[p]) {

// Check if process can finish

bool can\_finish = true;

for (int r = 0; r < R; r++) {

if (need[p][r] > work[r]) {

can\_finish = false;

break;

}

}

// If the process can finish, simulate completion

if (can\_finish) {

safe\_sequence[count++] = p;

for (int r = 0; r < R; r++) {

work[r] += allocation[p][r];

}

finish[p] = true;

found = true;

break;

}

}

}

// If no process can finish, the system is in an unsafe state

if (!found) {

printf("System is in an unsafe state!\n");

return false;

}

}

// If all processes can finish, the system is in a safe state

printf("System is in a safe state!\nSafe Sequence: ");

for (int i = 0; i < P; i++) {

printf("P%d ", safe\_sequence[i]);

}

printf("\n");

return true;

}

// Function to check if the request violates deadlock prevention rules (Hold and Wait)

bool check\_deadlock\_prevention(int request[], int available[], int allocation[]) {

for (int r = 0; r < R; r++) {

if (request[r] > available[r]) {

printf("Request exceeds available resources!\n");

return false;

}

}

// Check if the request is larger than what the process will need (deadlock prevention rule)

for (int r = 0; r < R; r++) {

if (request[r] > allocation[r]) {

printf("Process has already been allocated more resources than requested (Deadlock Prevention Violation)!\n");

return false;

}

}

// If everything is fine, return true

return true;

}

int main() {

int available[R] = {3, 3, 2}; // Available resources (e.g., 3 instances of resource 1, 3 instances of resource 2, and 2 instances of resource 3)

int maximum[P][R] = {

{7, 5, 3}, // Max demand for P0

{3, 2, 2}, // Max demand for P1

{9, 0, 2}, // Max demand for P2

{2, 2, 2}, // Max demand for P3

{4, 3, 3} // Max demand for P4

};

int allocation[P][R] = {

{0, 1, 0}, // P0's allocation

{2, 0, 0}, // P1's allocation

{3, 0, 2}, // P2's allocation

{2, 1, 1}, // P3's allocation

{0, 0, 2} // P4's allocation

};

int need[P][R];

// Calculate the need matrix (need = max - allocation)

for (int i = 0; i < P; i++) {

for (int j = 0; j < R; j++) {

need[i][j] = maximum[i][j] - allocation[i][j];

}

}

// Option to choose between deadlock avoidance and prevention

int choice;

printf("Choose an option:\n");

printf("1. Check Deadlock Avoidance (Banker's Algorithm)\n");

printf("2. Check Deadlock Prevention (Hold and Wait)\n");

printf("Enter your choice (1/2): ");

scanf("%d", &choice);

if (choice == 1) {

// Check for deadlock avoidance (safe state)

is\_safe\_state(available, maximum, allocation, need);

} else if (choice == 2) {

// Check for deadlock prevention (Hold and Wait)

int request[R];

printf("Enter the resource request for a process (e.g., 1 0 2): ");

for (int i = 0; i < R; i++) {

scanf("%d", &request[i]);

}

// Choose which process to check

int process\_id;

printf("Enter the process ID (0 to 4): ");

scanf("%d", &process\_id);

if (check\_deadlock\_prevention(request, available, allocation[process\_id])) {

printf("Request is valid under deadlock prevention rules.\n");

} else {

printf("Request violates deadlock prevention rules.\n");

}

} else {

printf("Invalid choice!\n");

}

return 0;

}

**OUTPUT:**

Choose an option:

1. Check Deadlock Avoidance (Banker's Algorithm)

2. Check Deadlock Prevention (Hold and Wait)

Enter your choice (1/2): 1

System is in a safe state!

Safe Sequence: P0 P1 P3 P4 P2

Choose an option:

1. Check Deadlock Avoidance (Banker's Algorithm)

2. Check Deadlock Prevention (Hold and Wait)

Enter your choice (1/2): 2

Enter the resource request for a process (e.g., 1 0 2): 1 0 1

Enter the process ID (0 to 4): 2

Request is valid under deadlock prevention rules.