**Slip 10 .**

**Q1 Write a program to illustrate the concept of orphan process (Using fork() and sleep())**

**ANS**

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/wait.h>

int main() {

pid\_t pid = fork(); // Create a new process

if (pid < 0) {

// Fork failed

perror("Fork failed");

exit(1);

}

if (pid == 0) {

// Child process

printf("Child Process: PID = %d, Parent PID = %d\n", getpid(), getppid());

sleep(5); // Sleep for 5 seconds to ensure parent terminates first

printf("Child Process: My parent has terminated. I am now an orphan process. PID = %d\n", getpid());

} else {

// Parent process

printf("Parent Process: PID = %d\n", getpid());

sleep(2); // Sleep for 2 seconds before terminating

printf("Parent Process: I am terminating now.\n");

exit(0); // Parent terminates

}

return 0;

}

**Q2: Write the simulation program to implement demand paging and show the page scheduling and total number of page faults for the following given page reference string. Give input n=3 as the number of memory frames. Reference String : 12,15,12,18,6,8,11,12,19,12,6,8,12,15,19,8 Implement OPT**

**Ans :**

#include <stdio.h>

#include <stdlib.h>

#define MAX\_FRAMES 10 // Maximum number of frames

// Function to find the index of the page that will not be used for the longest period

int findOptimalPage(int frames[], int num\_frames, int pages[], int n, int current\_index) {

int furthest\_use = -1;

int index\_to\_replace = -1;

for (int i = 0; i < num\_frames; i++) {

int j;

for (j = current\_index; j < n; j++) {

if (frames[i] == pages[j]) {

if (j > furthest\_use) {

furthest\_use = j;

index\_to\_replace = i;

}

break;

}

}

// If the page is not found in future references, we can replace it immediately

if (j == n) {

return i; // Return the index of the page that is not going to be used again

}

}

// If all pages are found in future references, replace the one that will be used furthest in the future

return index\_to\_replace;

}

// Function to implement OPT page replacement algorithm

void opt\_page\_replacement(int pages[], int n, int num\_frames) {

int frames[MAX\_FRAMES];

int page\_faults = 0;

// Initialize frames to -1 to indicate they are empty

for (int i = 0; i < num\_frames; i++) {

frames[i] = -1;

}

printf("Page Reference String: ");

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

printf("%d ", pages[i]);

}

printf("\n");

printf("Frame States:\n");

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

int is\_page\_present = 0;

// Check if the page is already in the frames

for (int j = 0; j < num\_frames; j++) {

if (frames[j] == pages[i]) {

is\_page\_present = 1; // Page is present

break;

}

}

if (!is\_page\_present) {

// Page fault occurs

page\_faults++;

int index\_to\_replace = findOptimalPage(frames, num\_frames, pages, n, i + 1);

frames[index\_to\_replace] = pages[i]; // Replace the page

}

// Print the current state of frames

printf("Current Frame State: ");

for (int j = 0; j < num\_frames; j++) {

if (frames[j] != -1) {

printf("%d ", frames[j]);

} else {

printf("X "); // 'X' indicates an empty frame

}

}

printf("\n");

}

printf("\nTotal Page Faults: %d\n", page\_faults);

}

int main() {

int n = 16; // Number of pages in reference string

int pages[] = {12, 15, 12, 18, 6, 8, 11, 12, 19, 12, 6, 8, 12, 15, 19, 8};

int num\_frames = 3; // Fixed number of memory frames

opt\_page\_replacement(pages, n, num\_frames);

return 0;

}

**Slip 11 .**

**Q.1 Create a child process using fork(), display parent and child process id. Child process will display the message “Hello World” and the parent process should display “Hi”. [**

**Ans:**

#include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

int main() {

pid\_t pid = fork(); // Create a child process

if (pid < 0) {

// Fork failed

perror("Fork failed");

return 1;

} else if (pid == 0) {

// This block is executed by the child process

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

printf("Hello World\n");

} else {

// This block is executed by the parent process

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

printf("Hi\n");

}

return 0;

}

**Q.2 Write the simulation program to implement demand paging and show the page scheduling and total number of page faults for the following given page reference string. Give input n as the number of memory frames. Reference String: 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1 Implement FIFO**

**ANS :**

#include <stdio.h>

#include <stdlib.h>

#define MAX\_FRAMES 10 // Maximum number of frames

// Function to implement FIFO page replacement algorithm

void fifo\_page\_replacement(int pages[], int n, int num\_frames) {

int frames[MAX\_FRAMES];

int page\_faults = 0;

int next\_frame = 0;

// Initialize frames to -1 to indicate they are empty

for (int i = 0; i < num\_frames; i++) {

frames[i] = -1;

}

printf("Page Reference String: ");

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

printf("%d ", pages[i]);

}

printf("\n");

printf("Frame States:\n");

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

int is\_page\_present = 0;

// Check if the page is already in the frames

for (int j = 0; j < num\_frames; j++) {

if (frames[j] == pages[i]) {

is\_page\_present = 1; // Page is present

break;

}

}

if (!is\_page\_present) {

// Page fault occurs

page\_faults++;

frames[next\_frame] = pages[i]; // Add new page into the frame

next\_frame = (next\_frame + 1) % num\_frames; // Move to the next frame in a circular manner

}

// Print the current state of frames

printf("Current Frame State: ");

for (int j = 0; j < num\_frames; j++) {

if (frames[j] != -1) {

printf("%d ", frames[j]);

} else {

printf("X "); // 'X' indicates an empty frame

}

}

printf("\n");

}

printf("\nTotal Page Faults: %d\n", page\_faults);

}

int main() {

int n = 12; // Number of pages in the reference string

int pages[] = {0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1};

int num\_frames;

printf("Enter the number of memory frames: ");

scanf("%d", &num\_frames);

if (num\_frames > MAX\_FRAMES || num\_frames <= 0) {

printf("Please enter a valid number of frames (1 to %d).\n", MAX\_FRAMES);

return 1;

}

fifo\_page\_replacement(pages, n, num\_frames);

return 0;

}

**Slip 12 ,**

**Q.1 [10] Write a program to illustrate the concept of orphan process ( Using fork() and sleep()) .**

**Ans:**

#include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

#include <stdlib.h>

#include <sys/wait.h>

int main() {

pid\_t pid = fork(); // Create a child process

if (pid < 0) {

// Fork failed

perror("Fork failed");

return 1;

} else if (pid == 0) {

// This block is executed by the child process

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

printf("Child is going to sleep for 5 seconds...\n");

sleep(5); // Child sleeps for 5 seconds

printf("Child Process %d is now awake!\n", getpid());

exit(0); // Child terminates

} else {

// This block is executed by the parent process

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

printf("Parent is terminating...\n");

exit(0); // Parent terminates immediately

}

return 0;

}

**Q.2 Write the simulation program to implement demand paging and show the page scheduling and total number of page faults for the following given page reference string. Give input n as the number of memory frames. Reference String : 12,15,12,18,6,8,11,12,19,12,6,8,12,15,19,8 Implement OPT**

**Ans:**

#include <stdio.h>

#include <stdlib.h>

#define MAX\_FRAMES 10 // Maximum number of frames

// Function to find the optimal page to replace

int findOptimalPage(int frames[], int frame\_count, int pages[], int current\_page\_index, int total\_pages) {

int farthest = -1; // To track the farthest page

int page\_to\_replace = -1;

for (int i = 0; i < frame\_count; i++) {

int j;

for (j = current\_page\_index; j < total\_pages; j++) {

if (frames[i] == pages[j]) {

if (j > farthest) {

farthest = j; // Update farthest usage index

page\_to\_replace = i; // Update page to replace

}

break;

}

}

// If the page is not found in the future references

if (j == total\_pages) {

return i; // Return the first unused page

}

}

return page\_to\_replace; // Return the page to replace based on optimal choice

}

// Function to implement OPT page replacement algorithm

void optimal\_page\_replacement(int pages[], int total\_pages, int frame\_count) {

int frames[MAX\_FRAMES];

int page\_faults = 0;

// Initialize frames to -1 to indicate they are empty

for (int i = 0; i < frame\_count; i++) {

frames[i] = -1;

}

printf("Page Reference String: ");

for (int i = 0; i < total\_pages; i++) {

printf("%d ", pages[i]);

}

printf("\n");

printf("Frame States:\n");

for (int i = 0; i < total\_pages; i++) {

int is\_page\_present = 0;

// Check if the page is already in the frames

for (int j = 0; j < frame\_count; j++) {

if (frames[j] == pages[i]) {

is\_page\_present = 1; // Page is present

break;

}

}

if (!is\_page\_present) {

// Page fault occurs

page\_faults++;

int replace\_index = findOptimalPage(frames, frame\_count, pages, i + 1, total\_pages);

frames[replace\_index] = pages[i]; // Replace page

}

// Print the current state of frames

printf("Current Frame State: ");

for (int j = 0; j < frame\_count; j++) {

if (frames[j] != -1) {

printf("%d ", frames[j]);

} else {

printf("X "); // 'X' indicates an empty frame

}

}

printf("\n");

}

printf("\nTotal Page Faults: %d\n", page\_faults);

}

int main() {

int total\_pages = 16; // Number of pages in the reference string

int pages[] = {12, 15, 12, 18, 6, 8, 11, 12, 19, 12, 6, 8, 12, 15, 19, 8};

int frame\_count;

printf("Enter the number of memory frames: ");

scanf("%d", &frame\_count);

if (frame\_count > MAX\_FRAMES || frame\_count <= 0) {

printf("Please enter a valid number of frames (1 to %d).\n", MAX\_FRAMES);

return 1;

}

optimal\_page\_replacement(pages, total\_pages, frame\_count);

return 0;

}

**Slip 13 .**

**Q.1 Write a program that demonstrates the use of nice() system call. After a child process is started using fork(), assign higher priority to the child using nice() system call.**

**Ans:**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/resource.h>

int main() {

pid\_t pid;

int nice\_value = 5; // Value to increase the priority of the child process

// Create a child process

pid = fork();

if (pid < 0) {

// Error in fork

perror("Fork failed");

exit(EXIT\_FAILURE);

} else if (pid == 0) {

// Child process

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

// Set higher priority by lowering the nice value

if (nice(-nice\_value) == -1) {

perror("Nice failed in child process");

exit(EXIT\_FAILURE);

}

// Display the new nice value of the child

int current\_nice = nice(0); // Get the current nice value

printf("Child process nice value set to: %d\n", current\_nice);

exit(EXIT\_SUCCESS);

} else {

// Parent process

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

printf("Child Process created with ID: %d\n", pid);

// Optionally wait for the child to finish

wait(NULL);

}

return 0;

}

**Q 2: Write the simulation program using SJF(non-preemptive). The arrival time and first CPU bursts of different jobs should be input to the system. The Assume the fixed I/O waiting time (2 units).The next CPU burst should be generated using random function. The output should give the Gantt chart, Turnaround Time and Waiting time for each process and average times.**

**Ans:**

#include <stdio.h>

#include <stdlib.h>

#define MAX\_PROCESSES 100

typedef struct {

int process\_id;

int arrival\_time;

int burst\_time;

int waiting\_time;

int turnaround\_time;

} Process;

// Function to compare two processes based on their burst time

int compare(const void \*a, const void \*b) {

Process \*p1 = (Process \*)a;

Process \*p2 = (Process \*)b;

return (p1->burst\_time - p2->burst\_time);

}

// Function to calculate waiting and turnaround times

void calculateTimes(Process processes[], int n) {

int total\_waiting\_time = 0, total\_turnaround\_time = 0;

processes[0].waiting\_time = 0; // First process has no waiting time

for (int i = 1; i < n; i++) {

processes[i].waiting\_time = processes[i - 1].waiting\_time + processes[i - 1].burst\_time + 2; // Adding I/O wait time

total\_waiting\_time += processes[i].waiting\_time;

}

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

processes[i].turnaround\_time = processes[i].waiting\_time + processes[i].burst\_time;

total\_turnaround\_time += processes[i].turnaround\_time;

}

printf("\nAverage Waiting Time: %.2f\n", (float)total\_waiting\_time / n);

printf("Average Turnaround Time: %.2f\n", (float)total\_turnaround\_time / n);

}

// Function to print the Gantt chart

void printGanttChart(Process processes[], int n) {

printf("\nGantt Chart:\n");

printf("|");

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

printf(" P%d |", processes[i].process\_id);

}

printf("\n");

int current\_time = 0;

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

current\_time += processes[i].burst\_time + 2; // Adding I/O wait time

printf("%-3d ", current\_time);

}

printf("\n");

}

int main() {

int n;

Process processes[MAX\_PROCESSES];

// Input number of processes

printf("Enter the number of processes: ");

scanf("%d", &n);

// Input arrival times and burst times for each process

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

processes[i].process\_id = i + 1; // Assigning process IDs

printf("Enter arrival time and burst time for process P%d: ", i + 1);

scanf("%d %d", &processes[i].arrival\_time, &processes[i].burst\_time);

}

// Sort processes by arrival time

qsort(processes, n, sizeof(Process), compare);

// Scheduling

int current\_time = 0;

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

if (current\_time < processes[i].arrival\_time) {

current\_time = processes[i].arrival\_time; // Wait until the process arrives

}

processes[i].waiting\_time = current\_time - processes[i].arrival\_time;

current\_time += processes[i].burst\_time + 2; // Adding I/O wait time

processes[i].turnaround\_time = processes[i].waiting\_time + processes[i].burst\_time;

}

// Print results

printf("\nProcess\tArrival Time\tBurst Time\tWaiting Time\tTurnaround Time\n");

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

printf("P%d\t%d\t\t%d\t\t%d\t\t%d\n", processes[i].process\_id,

processes[i].arrival\_time, processes[i].burst\_time,

processes[i].waiting\_time, processes[i].turnaround\_time);

}

// Print Gantt chart

printGanttChart(processes, n);

// Calculate average times

calculateTimes(processes, n);

return 0;

}

**Slip 14**  .

**Q.1 Write a program to find the execution time taken for execution of a given set of instructions (use clock() function)**

**Ans:**

#include <stdio.h>

#include <time.h>

void performTask() {

// Simulate some task

long long sum = 0;

for (long long i = 0; i < 100000000; i++) {

sum += i; // Some computation

}

printf("Sum: %lld\n", sum);

}

int main() {

// Record the start time

clock\_t start\_time = clock();

// Execute the task

performTask();

// Record the end time

clock\_t end\_time = clock();

// Calculate the elapsed time in seconds

double time\_taken = (double)(end\_time - start\_time) / CLOCKS\_PER\_SEC;

// Print the execution time

printf("Execution Time: %.6f seconds\n", time\_taken);

return 0;

}

**Q.2 Write the simulation program to implement demand paging and show the page scheduling and total number of page faults for the following given page reference string. Give input n =3 as the number of memory frames. Reference String : 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1 Implement FIFO**

**Ans:**

#include <stdio.h>

#include <stdlib.h>

#define MAX\_FRAMES 10 // Maximum number of frames allowed

int isPageInFrames(int frames[], int numFrames, int page) {

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

if (frames[i] == page) {

return 1; // Page found in frames

}

}

return 0; // Page not found

}

void displayFrames(int frames[], int numFrames) {

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

if (frames[i] != -1) {

printf("%d ", frames[i]);

} else {

printf("- ");

}

}

printf("\n");

}

int main() {

int frames[MAX\_FRAMES], numFrames, pageReference[] = {0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1};

int numPages = sizeof(pageReference) / sizeof(pageReference[0]);

int pageFaults = 0, nextFrameIndex = 0;

// Input number of memory frames

numFrames = 3; // Fixed value as per the requirement

// Initialize frames

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

frames[i] = -1; // -1 indicates an empty frame

}

printf("Page Reference String: ");

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

printf("%d ", pageReference[i]);

}

printf("\n\n");

printf("FIFO Page Replacement:\n");

printf("Frames: ");

displayFrames(frames, numFrames);

// Process each page reference

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

int currentPage = pageReference[i];

// Check if the page is already in frames

if (!isPageInFrames(frames, numFrames, currentPage)) {

// Page fault occurs

pageFaults++;

printf("Page Fault: %d\n", currentPage);

// Replace the oldest page in the FIFO manner

frames[nextFrameIndex] = currentPage;

nextFrameIndex = (nextFrameIndex + 1) % numFrames; // Circular increment

// Display current frames

displayFrames(frames, numFrames);

} else {

printf("No Page Fault: %d\n", currentPage);

}

}

// Display total number of page faults

printf("\nTotal Page Faults: %d\n", pageFaults);

return 0;

}

**Slip 15 .**

**Q.1 Write a program to create a child process using fork().The parent should goto sleep state and child process should begin its execution. In the child process, use execl() to execute the “ls” command.**

**Ans:**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

int main() {

pid\_t pid; // Variable to hold process ID

// Create a child process

pid = fork();

if (pid < 0) {

// Fork failed

perror("Fork failed");

exit(1);

} else if (pid == 0) {

// This block is executed by the child process

printf("Child Process (PID: %d) executing 'ls' command...\n", getpid());

// Replace the child process with the 'ls' command

execl("/bin/ls", "ls", NULL);

// If execl returns, there was an error

perror("execl failed");

exit(1);

} else {

// This block is executed by the parent process

printf("Parent Process (PID: %d) going to sleep...\n", getpid());

// Parent goes to sleep for 5 seconds

sleep(5);

// After sleeping, the parent can perform other tasks or exit

printf("Parent Process (PID: %d) woke up!\n", getpid());

}

return 0;

}

**Q.2 Write the simulation program to implement demand paging and show the page scheduling and total number of page faults for the following given page reference string. Give input n as the number of memory frames. Reference String :7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2 Implement LRU**

**Ans:**

#include <stdio.h>

#include <stdlib.h>

#define MAX\_FRAMES 10

#define MAX\_PAGES 100

// Function to find the index of the least recently used page

int findLRU(int time[], int n) {

int minimum = time[0], pos = 0;

for (int i = 1; i < n; ++i) {

if (time[i] < minimum) {

minimum = time[i];

pos = i;

}

}

return pos;

}

// Function to simulate LRU page replacement

void lruPageReplacement(int pages[], int n, int frames) {

int frame[MAX\_FRAMES], time[MAX\_FRAMES];

int pageFaults = 0;

// Initialize frames and time arrays

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

frame[i] = -1; // Indicates an empty frame

time[i] = 0; // Initialize time for LRU tracking

}

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

int j;

int pageFound = 0;

// Check if the page is already in one of the frames

for (j = 0; j < frames; ++j) {

if (frame[j] == pages[i]) {

pageFound = 1; // Page is found in frame

time[j] = i; // Update time of the page

break;

}

}

// If page is not found in any frame

if (!pageFound) {

int lruIndex = findLRU(time, frames); // Find the LRU page

frame[lruIndex] = pages[i]; // Replace it with the new page

time[lruIndex] = i; // Update the time

pageFaults++;

printf("Page Fault! Loaded page: %d\n", pages[i]);

}

// Print current state of frames

printf("Current Frames: ");

for (j = 0; j < frames; ++j) {

if (frame[j] != -1)

printf("%d ", frame[j]);

else

printf("\_ "); // Indicate empty frames

}

printf("\n");

}

printf("Total Page Faults: %d\n", pageFaults);

}

int main() {

int pages[] = {7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2}; // Page reference string

int n = sizeof(pages) / sizeof(pages[0]); // Number of pages

int frames;

// Input number of memory frames

printf("Enter the number of memory frames (max %d): ", MAX\_FRAMES);

scanf("%d", &frames);

if (frames > MAX\_FRAMES || frames <= 0) {

printf("Invalid number of frames! Exiting.\n");

return 1;

}

lruPageReplacement(pages, n, frames); // Run LRU page replacement simulation

return 0;

}