Lab 2: Process control system calls & CPU Scheduling

Goal: This lab will deepen your understanding of various concepts of process management in Linux, specifically on system calls such as FORK(), EXEC(), WAIT(), and EXIT(). This will also improve your understanding of various non-preemptive & preemptive CPU scheduling such as FCFS, SJF, SRTF, Round-Robin, etc. (discussed in the class).

You need to write a simple program using these system calls, as a warmup to solving this lab exercise. This is the demonstration of FORK(), EXEC(), WAIT(), and EXIT() system calls along with zombie and orphan states.

Before you begin

- Familiarize yourself with the various process related system calls in Linux: FORK(), EXEC(), WAIT(), and EXIT(). The "man pages" in Linux are a good source of learning. You can access the man pages from the Linux terminal by typing man fork, man 2 fork and so on.

- It is important to understand the different variants of these system calls. In particular, there are many different variants of the exec and wait system calls; you need to understand these to use the correct variant in your code. For example, you may need to invoke wait differently depending on whether you need to block for a child to terminate or not.

- Familiarize yourself with simple shell commands in Linux like echo, cat, sleep, ls, ps, top, grep and so on. To implement these commands in your shell, you must simply "exec" these existing executables, and not implement the functionality yourself.

- Familiarize yourself with various CPU scheduling algorithms like FIRST COME FIRST SERVE (FCFS), SHORTEST JOB FIRST (SJF), SHORTEST REMAINING TIME FIRST (SRTF)/PREEMPTIVE SJF, ROUND-ROBIN.

Exercises

1. Write a C program to create child processes of a main process using fork system call for N times. N can be any number ranging from 1 to 10. Also print a message with each process saying "Hello I am a child process\ parent process. My Pid is ="__". Here, the child process should print “I am a child process” and exit. The parent should wait to reap the child, print a message “I am a parent process” after reaping the child, and then exit. Analyze the number of child processes while varying the value of N & mention in the report.

   (**Note** - Use getpid() to get the Pid of a process)

2. Write a C program in which main program accepts the integers to be sorted. Main program uses the FORK system call to create a new process called a child process. Parent process sorts the integers using sorting algorithm and waits for child process using WAIT system call to sort the integers using any sorting algorithm. Also demonstrate zombie and orphan states.

3. Write a C program in which main program accepts an integer array. Main program uses the FORK system call to create a new process called a child process. Parent process sorts an integer array and passes the sorted array to child process through the command line arguments of EXEC system call. The child process uses EXEC system call to load new program that uses this sorted array for performing the binary search to search the particular item in the array.

4. Write a c program to simulate the FCFS scheduling algorithm.

   Hint: To calculate the average waiting time using the FCFS algorithm first the waiting time of the first process is kept zero and the waiting time of the second process is the burst time of the first process and the waiting time of the third process is the sum of the burst times of the first and the second process and so on. After calculating all the waiting times the average waiting time is calculated as the average of all the waiting times. FCFS mainly says first come first serve the algorithm which came first will be served first.

   ALGORITHM:

   Step 1: Start the process

   Step 2: Accept the number of processes in the ready Queue

   Step 3: For each process in the ready Q, assign the process name and the burst time

   Step 4: Set the waiting of the first process as =0‘and its burst time as its turnaround time

Step 5: for each process in the Ready Q calculate

a). Waiting time (n) = waiting time (n-1) + Burst time (n-1)

b). Turnaround time (n)= waiting time(n)+Burst time(n)

Step 6: Calculate

a) Average waiting time = Total waiting Time / Number of process

b) Average Turnaround time = Total Turnaround Time / Number of process

Step 7: Stop the process

5. Write a program in C to stimulate the CPU scheduling algorithm Shortest job first (Non-Preemption)

   Hint: To calculate the average waiting time in the shortest job first algorithm the sorting of the process based on their burst time in ascending order then calculate the waiting time of each process as the sum of the bursting times of all the process previous or before to that process.

ALGORITHM:
Step 1: Start the process
Step 2: Accept the number of processes in the ready Queue
Step 3: For each process in the ready Q, assign the process id and accept the CPU burst time
Step 4: Start the Ready Q according the shortest Burst time by sorting according to lowest to highest burst time.
Step 5: Set the waiting time of the first process as _0' and its turnaround time as its burst time.
Step 6: Sort the processes names based on their Burt time
Step 7: For each process in the ready queue,
calculate
a) Waiting time(n)= waiting time (n-1) + Burst time (n-1)
b) Turnaround time (n)= waiting time(n)+Burst time(n)
Step 8: Calculate
c) Average waiting time = Total waiting Time / Number of process
d)Average Turnaround time = Total Turnaround Time / Number of process
Step 9: Stop the process

6. Write a C program to simulate the Shortest Remaining Time First scheduling algorithm.

7. Write a C program to simulate the CPU scheduling algorithm round-robin.

Hint: To aim is to calculate the average waiting time. There will be a time slice, each process should be executed within that time-slice and if not it will go to the waiting state so first check whether the burst time is less than the time-slice. If it is less than it assign the waiting time to the sum of the total times. If it is greater than the burst-time then subtract the time slot from the actual

burst time and increment it by time-slot and the loop continues until all the processes are completed.

ALGORITHM:
Step 1: Start the process
Step 2: Accept the number of processes in the ready Queue and time quantum (or) time slice
Step 3: For each process in the ready Q, assign the process id and accept the CPU burst time
Step 4: Calculate the no. of time slices for each process where No. of time slice for process (n) = burst time process (n)/time slice
Step 5: If the burst time is less than the time slice then the no. of time slices =1.
Step 6: Consider the ready queue is a circular Q, calculate
a) Waiting time for process (n) = waiting time of process(n-1)+ burst time of process(n-1) + the time difference in getting the CPU fromprocess(n-1)
b) Turnaround time for process(n) = waiting time of process(n) + burst time of process(n)+ the time difference in getting CPU from process(n).
Step 7: Calculate
c) Average waiting time = Total waiting Time / Number of process
d) Average Turnaround time = Total Turnaround Time / Number ofprocess
Step 8: Stop the process

Submission Format:- You have to upload: (1) The source code in the following format: Assgn2Src-<RollNo>.c (2) Readme: Assgn2Readme-<RollNo>.txt, which contains the instructions for executing the program. (3) Report: Assgn2Report-<RollNo>.pdf. Name the zipped document as: Assgn2-<RollNo>.zip.

Note: Please follow this naming convention mentioned above.

Grading Policy:- The policy for grading this assignment will be - (1) Design as described in the report and analysis of the results: 50% (2) Execution of the tasks based on the description in the readme: 40% (3) Code documentation and indentation: 10%.

Please note:
- All assignments for this course have a late submission policy of a penalty of 10% each day after the deadline of six days. After that, it will not be evaluated.
- All submissions are subject to plagiarism checks. Any case of plagiarism will be dealt with severely.