Read the following to understand the parent & child process concept.

Process state Diagram

The parent & child process concept

Understand orphan processes

Understand zombie processes

what is getpid() and how to use it..??

what is getppid() and how to use it..??

what is fork()? what happens when you use fork() in your program..??

what is wait() / waitpid() and how to use it..??

what are zombie processes? when they will get created.? How to kill the zombie processes..?

what are orphan ​processes? when they will get created.?

what is user space?

what is kernel space / system space?

What is IPC(inter process communication) and what is the need of it.

what is Unnamed PIPE? how to create it? When we need this??

what is named PIPE? how to create it? When we need this??

What is the difference between Unnamed PIPE and named PIPE?

what are execl,exec,execv and difference between them..??

what is semaphore? Explore sem\_init(), sem\_create(), sem\_destroy(), sem\_unlink(), sem\_wait(), sem\_post().

what is mutex? Difference between semaphore and mutex.

Process Management

Note :

1. Read Man pages of the functions/commands mentioned in the hint of each question.

2. You will get the name of header file to be included for that function to work, in the man page of that respective function.

Questions :

1. Parent and child process

Write a C/CPP program to create a child process. Child should print its pid and its parent’s pid and should exit by printing message as “Child Exiting …”. Parent should print its pid and should exit by printing message as “Parent Exiting ..”.

(Hint(functions to be used) : getpid, fork, getppid)

ANS:

#include <iostream>

#include <sys/types.h>

#include <unistd.h>

#include <stdlib.h>

using namespace std;

int main() {

pid\_t pid;

cout << "Before fork!!!!!" << endl;

pid = fork();

if(pid==0) {

cout <<"this is child process:" << getpid() <<" " << getppid() << endl;

execv("/bin/date", "date", NULL);

cout << endl << "-----------------------------" << endl;

exit(0);

exit(0);

} else if(pid>0) {

cout << "this is parent process:" << getpid()<< " " << getppid() << endl;

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

exit(0);

}

else {

cout << "process is not created" << endl;

}

exit(0);

return 0;

}

2. Scheduling functions

Write a program in C/CPP to check the scheduling policy used by the process and its priority.

(Hint(functions to be used) : sched\_getscheduler, getpid)

ANS:

#include <iostream>

#include <sys/types.h>

#include <unistd.h>

#include <stdlib.h>

#include <sched.h>

using namespace std;

int main()

{

pid\_t pid;

cout << "Before fork!!!!!" << endl;

pid = fork();

if (pid == 0)

{

// cout << pid << endl;

cout << "This is child process. Process ID: " << getpid() << " "

<< "Parent ID: " << getppid() << " "

<< "Scheduling Policy Status: "

<< sched\_getscheduler(pid) << endl;

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

cout << endl

<< "-----------------------------" << endl;

exit(0);

}

else if (pid > 0)

{

// cout << pid << endl;

cout << "this is parent process:" << getpid() << " "

<< "Parent ID: " << getppid() << " "

<< "Scheduling Policy Status: "

<< sched\_getscheduler(pid) << endl;

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

exit(0);

}

else

{

cout << "process is not created" << endl;

}

exit(0);

return 0;

}

3. Scheduling functions

Write a program in C/CPP to get the current scheduling policy of the process. The program should change

the scheduling policy to the other than current one.

Program should report errors if it fails to set the new scheduling policy.

(Hint(functions to be used) : sched\_setscheduler, getpid)

4. Scheduling algorithm

Write a program in C/CPP to take process name, its arrival time and execution/burst time as input.

Use FCFS(non-preemptive) algorithm to calculate wait time of each process, average

wait time, turnaround time of each process and average turnaround time.

Calculation of time will start from the arrival time of first process.

execution/burst time - Time required for execution of process

Wait time of process = response time of process - arrival time process

Response time of process : time at which process is scheduled to run

Average wait time = (sum of wait time of each process) / (number of processes)

Turnaround time of process = (finish/completion time of process) - (arrival time of process)

Average turnaround time = (sum of turnaround time of each process) / (number of processes)

ANS:

#include <iostream>

#include <bits/stdc++.h>

#include <iomanip>

using namespace std;

class Process

{

public:

string pName;

int aTime;

int bTime;

int rTime;

int cTime;

int wTime;

int tTime;

Process() {}

// Process(string pName, int aTime, int bTime)

// {

// this->pName = pName;

// this->aTime = aTime;

// this->bTime = bTime;

// }

};

bool arrTime(Process p1, Process p2)

{

return (p1.aTime < p2.aTime);

}

int main()

{

int num;

cout << "Enter Number of Processes: ";

cin >> num;

Process ps[num];

float avgwTime = 0;

float avgtTime = 0;

for (int i = 0; i < num; i++)

{

int at, bt;

string p;

cout << "Enter Process Name, Arrival Time and Burst Time: ";

cin >> p >> at >> bt;

ps[i].aTime = at;

ps[i].bTime = bt;

ps[i].pName = p;

}

// cout << "Hello Ritesh!!" << endl;

sort(ps, ps + num, arrTime);

for (int i = 0; i < num; i++)

{

cout << ps[i].pName << " "

<< ps[i].bTime << " "

<< ps[i].aTime << endl;

}

for (int i = 0; i < num; i++)

{

if (i == 0)

{

ps[i].rTime = ps[i].aTime;

}

else

{

ps[i].rTime = ps[i - 1].cTime;

}

ps[i].cTime = ps[i].rTime + ps[i].bTime;

ps[i].wTime = ps[i].rTime - ps[i].aTime;

ps[i].tTime = ps[i].cTime - ps[i].aTime;

avgwTime += ps[i].wTime;

avgtTime += ps[i].tTime;

}

avgwTime /= float(num);

avgtTime /= float(num);

cout << "AVG Wait Time: " << fixed << setprecision(2) << avgwTime << endl;

cout << "AVG Turnaround Time: " << fixed << setprecision(2) << avgtTime << endl;

return 0;

}