**Week 1: Demonstration of system calls**

1. **Write a program to create a child process using system call fork().**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

int main() {

printf("\nName - Anshuman Bhandari\nSection - A2\nRoll No.- 18\n\n");

int pid = fork();

if (pid < 0) {

perror("Fork failed");

exit(1);

}

else if (pid == 0)

printf("This is the child process with PID: %d\n", getpid());

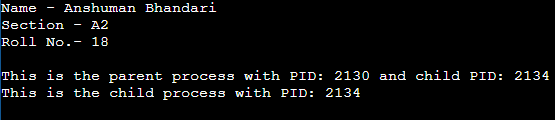
else

printf("This is the parent process with PID: %d and child PID: %d\n", getpid(), pid);

return 0;

}

**OUTPUT:**

****

1. **Write a program to print process Id's of parent and child process i.e. parent should print its own and its child process id while child process should print its own and its parent process id. (use getpid(), getppid())**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

int main() {

printf("\nName - Anshuman Bhandari\nSection - A2\nRoll No.- 18\n\n");

int pid = fork();

if (pid< 0) {

perror("Fork failed");

exit(1);

}

else if (pid == 0)

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

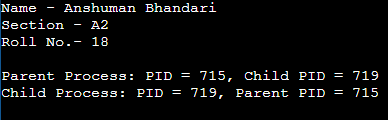
else

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

return 0;

}

**OUTPUT:**

****

1. **Write a program to create child process which will list all the files present in your system. Make sure that parent process waits until child has not completed its execution. (use wait(), exit()) What will happen if parent process dies before child process? Illustrate it by creating one more child of parent process.**

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <unistd.h>

int main() {

printf("\nName - Anshuman Bhandari\nSection - A2\nRoll No.- 18\n\n");

int pid;

pid = fork();

if (pid< 0) {

printf("Fork failed");

exit(1);

}

else if (pid == 0) {

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

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

perror("execl failed");

}

else {

printf("Parent process: Waiting for child to complete...\n");

int status;

wait(&status);

printf("Parent process: Child has completed.\n");

if (WIFEXITED(status))

printf("Child exited with status %d\n", WEXITSTATUS(status));

else

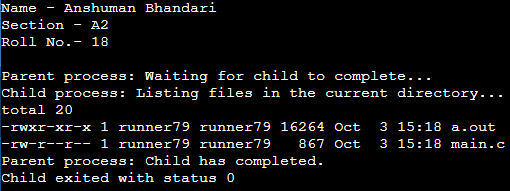
printf("Child did not exit successfully\n");

}

return 0;

}

**OUTPUT:**

****

1. **What will happen if parent process dies before child process? Illustrate it by creating one more child of parent process.**

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <unistd.h>

int main() {

printf("\nName - Anshuman Bhandari\nSection - A2\nRoll No.- 18\n\n");

pid\_t pid1, pid2;

pid1 = fork();

if (pid1 < 0)

perror("Fork failed for child 1");

else if (pid1 == 0) {

printf("First child process (PID: %d): Starting...\n", getpid());

sleep(2);

printf("First child process (PID: %d): Exiting...\n", getpid());

}

else {

pid2 = fork();

if (pid2 < 0) {

perror("Fork failed for child 2");

exit(1);

}

else if (pid2 == 0) {

printf("Second child process (PID: %d): Starting...\n", getpid());

sleep(5);

printf("Second child process (PID: %d): Exiting...\n", getpid());

exit(0);

}

else {

printf("Parent process (PID: %d): Exiting before second child finishes...\n", getpid());

exit(0);

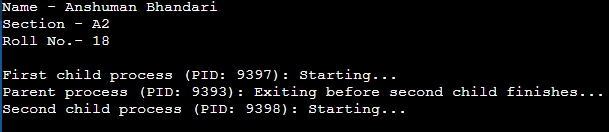
}

}

return 0;

}

**OUTPUT:**

****

**Week 2: Demonstration of system calls**

1. **Write a program to open a directory and list its contents. (use opendir(), readdir(), closedir() )**

#include <stdio.h>

#include <dirent.h>

int main() {

printf("\nName - Anshuman Bhandari\nSection - A2\nRoll No.- 18\n\n");

DIR \*dir;

struct dirent \*entry;

dir = opendir(".");

if (dir == NULL) {

perror("Can't open directory!\n");

return 1;

}

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

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

closedir(dir);

return 0;

}

**OUTPUT:**

1. **Write a program to show working of execlp() system call by executing ls command.**

#include <stdio.h>

#include <unistd.h>

int main()

{

printf("\nName - Anshuman Bhandari\nSection - A2\nRoll No.- 18\n\n");

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

perror("execlp terminated!");

return 1;

}

**OUTPUT:**

1. **Write a program to read a file and store your details in that file. Your program should also create one more file and store your friends details in that file. Once both files are created, print lines which are matching in both files.**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <fcntl.h>

#include <unistd.h>

#define BUFFER\_SIZE 256

void write\_details(const char \*filename, const char \*details) {

int fd = open(filename, O\_WRONLY | O\_CREAT | O\_TRUNC, 0644);

if (fd < 0) {

perror("open");

exit(1);

}

write(fd, details, strlen(details));

close(fd);

}

void read\_file(const char \*filename) {

char buffer[BUFFER\_SIZE];

int fd = open(filename, O\_RDONLY);

if (fd < 0) {

perror("open");

exit(1);

}

printf("Contents of %s:\n", filename);

while (read(fd, buffer, sizeof(buffer) - 1) > 0) {

buffer[BUFFER\_SIZE - 1] = '\0';

printf("%s", buffer);

}

close(fd);

}

int main() {

printf("\nName - Anshuman Bhandari\nSection - A2\nRoll No.- 18\n\n");

const char \*my\_details = "Name: Anshuman Bhandari\nAge: 20\nCity: Rishikesh\n";

const char \*friend\_details = "Name: Aman Singh\nAge: 20\nCity: Dehradun\n";

write\_details("my\_details.txt", my\_details);

write\_details("friend\_details.txt", friend\_details);

FILE \*my\_file = fopen("my\_details.txt", "r");

FILE \*friend\_file = fopen("friend\_details.txt", "r");

if (!my\_file || !friend\_file) {

perror("fopen FAILED!");

exit(1);

}

char my\_line[BUFFER\_SIZE];

while (fgets(my\_line, sizeof(my\_line), my\_file)) {

rewind(friend\_file);

char friend\_line[BUFFER\_SIZE];

while (fgets(friend\_line, sizeof(friend\_line), friend\_file)) {

if (strcmp(my\_line, friend\_line) == 0)

printf("Matching line: %s", my\_line);

}

}

fclose(my\_file);

fclose(friend\_file);

return 0;

}

**Week 3: Process Scheduling**

* 1. **FCFS – First Come First Served : process which arrives first will get the CPU first.**

#include <stdio.h>

#include <stdlib.h>

typedef struct process {

int at, bt, st, ct, tat, wt, rt, pid;

} st;

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

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

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

return p1->at > p2->at;

}

int comp1(const void \*a, const void \*b) {

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

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

return p1->pid > p2->pid;

}

void averageTime(st \*arr, int n) {

qsort(arr, n, sizeof(st), comp1);

float total\_wt = 0, total\_tat = 0, total\_rt = 0, total\_it = 0;

printf("\nPid\tAT\tBT\tCT\tTAT\tWT\tRT\n");

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

total\_wt += arr[i].wt;

total\_tat += arr[i].tat;

total\_rt += arr[i].rt;

total\_it += (i == 0) ? arr[i].at : (arr[i].st - arr[i - 1].ct);

printf("%d\t%d\t%d\t%d\t%d\t%d\t%d\n", arr[i].pid, arr[i].at, arr[i].bt, arr[i].ct, arr[i].tat, arr[i].wt, arr[i].rt);

}

printf("\nAverage Waiting time %f\n", total\_wt / (float)n);

printf("Average Turn Around time %f\n", total\_tat / (float)n);

printf("Average Response time %f\n", total\_rt / (float)n);

printf("CPU utilization %f\n", ((arr[n - 1].ct - total\_it) / (float)arr[n - 1].ct) \* 100);

printf("Throughput %f\n", (float)n / (arr[n - 1].ct - arr[0].at));

}

int main()

{

printf("\nName - Anshuman Bhandari\nSection - A2\nRoll No.- 18\n\n");

int n;

printf("Enter the Number of elements : ");

scanf("%d", &n);

st arr[n];

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

{

printf("For process %d Enter Arrival time and Burst time respectively : ", i + 1);

arr[i].pid = i + 1;

scanf("%d%d", &arr[i].at, &arr[i].bt);

}

qsort(arr, n, sizeof(st), comp);

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

if (i == 0) {

arr[0].ct = arr[0].at + arr[0].bt;

arr[0].st = arr[0].at;

}

arr[i].st = (arr[i - 1].ct >= arr[i].at) ? arr[i - 1].ct : arr[i].at;

if (arr[i].at >= arr[i - 1].ct) {

arr[i].ct = arr[i].at + arr[i].bt;

}

else {

arr[i].ct = arr[i - 1].ct + arr[i].bt;

}

arr[i].rt = arr[i].st - arr[i].at;

arr[i].tat = arr[i].ct - arr[i].at;

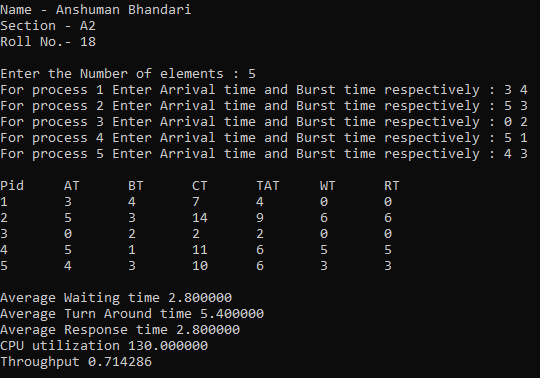
arr[i].wt = arr[i].tat - arr[i].bt;

}

averageTime(arr, n);

}

**OUTPUT:**

****

* 1. **SJF NP – Shortest Job First Non-Preemptive : process which needs CPU for least amount will get the CPU first. Here non-preemptive means currently running process leaves CPU voultarily after completing its execution.**

#include <stdio.h>

#include <stdlib.h>

#include <limits.h>

typedef struct process {

int at, bt, st, ct, tat, wt, rt, pid, is\_completed;

} st;

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

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

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

return p1->at > p2->at;

}

void averageTime(st \*arr, int n) {

float total\_wt = 0, total\_tat = 0, total\_rt = 0;

printf("\nPid\tAT\tBT\tCT\tTAT\tWT\tRT\n");

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

total\_wt += arr[i].wt;

total\_tat += arr[i].tat;

total\_rt += arr[i].rt;

printf("%d\t%d\t%d\t%d\t%d\t%d\t%d\n", arr[i].pid, arr[i].at, arr[i].bt, arr[i].ct, arr[i].tat, arr[i].wt, arr[i].rt);

}

printf("\nAverage Waiting time %f\n", total\_wt / (float)n);

printf("Average Turn Around time %f\n", total\_tat / (float)n);

printf("Average Response time %f\n", total\_rt / (float)n);

printf("CPU utilization %f\n", ((arr[n - 1].ct - arr[0].at) / (float)arr[n - 1].ct) \* 100);

printf("Throughput %f\n", (float)n / (arr[n - 1].ct - arr[0].at));

}

int main() {

printf("\nName - Anshuman Bhandari\nSection - A2\nRoll No.- 18\n\n");

int n;

printf("Enter the Number of elements : ");

scanf("%d", &n);

st arr[n];

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

printf("For process %d Enter Arrival time and Burst time respectively : ", i + 1);

arr[i].pid = i + 1;

scanf("%d%d", &arr[i].at, &arr[i].bt);

arr[i].is\_completed = 0;

}

int current\_time = 0, completed = 0, prev = 0;

qsort(arr, n, sizeof(st), comp);

while (completed != n) {

int idx = -1, mn = INT\_MAX;

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

if (arr[i].at <= current\_time && arr[i].is\_completed == 0) {

if (arr[i].bt < mn) {

mn = arr[i].bt;

idx = i;

}

if (arr[i].bt == mn) {

if (arr[i].at < arr[idx].at) {

mn = arr[i].bt;

idx = i;

}

}

}

}

if (idx != -1) {

arr[idx].st = current\_time;

arr[idx].ct = current\_time + arr[idx].bt;

arr[idx].tat = arr[idx].ct - arr[idx].at;

arr[idx].wt = arr[idx].tat - arr[idx].bt;

arr[idx].rt = arr[idx].st - arr[idx].at;

arr[idx].is\_completed = 1;

completed++;

current\_time = arr[idx].ct;

} else {

current\_time++;

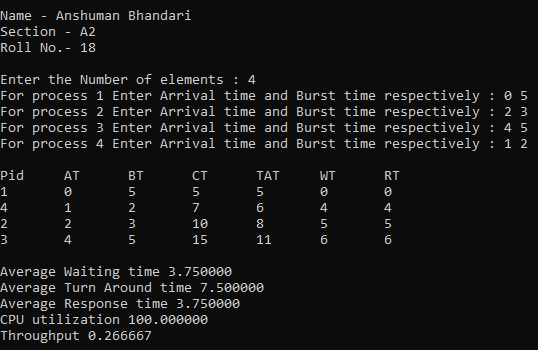
}

}

averageTime(arr, n);

}

**OUTPUT:**

****

* 1. **SJF P – Shortest Job First Preemptive – Here preemptive means operating system decides when to move currently running process.**

#include <stdio.h>

#include <stdlib.h>

#include <limits.h>

typedef struct process {

int at, bt, st, ct, tat, wt, rt, pid, bt\_remaining, is\_completed;

} st;

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

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

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

return p1->at > p2->at;

}

void averageTime(st \*arr, int n, float total\_it) {

float total\_wt = 0, total\_tat = 0, total\_rt = 0;

printf("\nPid\tAT\tBT\tCT\tTAT\tWT\tRT\n");

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

total\_wt += arr[i].wt;

total\_tat += arr[i].tat;

total\_rt += arr[i].rt;

printf("%d\t%d\t%d\t%d\t%d\t%d\t%d\n", arr[i].pid, arr[i].at, arr[i].bt, arr[i].ct, arr[i].tat, arr[i].wt, arr[i].rt);

}

printf("\nAverage Waiting time %f\n", total\_wt / (float)n);

printf("Average Turn Around time %f\n", total\_tat / (float)n);

printf("Average Response time %f\n", total\_rt / (float)n);

printf("CPU utilization %f\n", ((arr[n - 1].ct - total\_it) / (float)arr[n - 1].ct) \* 100);

printf("Throughput %f\n", (float)n / (arr[n - 1].ct - arr[0].at));

}

int main() {

printf("\nName - Anshuman Bhandari\nSection - A2\nRoll No.- 18\n\n");

int n;

printf("Enter the Number of elements : ");

scanf("%d", &n);

st arr[n];

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

printf("For process %d Enter Arrival time and Burst time respectively : ", i + 1);

arr[i].pid = i + 1;

scanf("%d%d", &arr[i].at, &arr[i].bt);

arr[i].bt\_remaining = arr[i].bt;

arr[i].is\_completed = 0;

}

int current\_time = 0, completed = 0, total\_it = 0, prev = 0;

qsort(arr, n, sizeof(st), comp);

while (completed != n) {

int idx = -1, mn = INT\_MAX;

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

if (arr[i].at <= current\_time && arr[i].is\_completed == 0) {

if (arr[i].bt\_remaining < mn) {

mn = arr[i].bt\_remaining;

idx = i;

}

if (arr[i].bt\_remaining == mn) {

if (arr[i].at < arr[idx].at) {

mn = arr[i].bt\_remaining;

idx = i;

}

}

}

}

if (idx != -1) {

if (arr[idx].bt\_remaining == arr[idx].bt) {

arr[idx].st = current\_time;

total\_it += arr[idx].st - prev;

}

arr[idx].bt\_remaining -= 1;

current\_time++;

prev = current\_time;

if (arr[idx].bt\_remaining == 0) {

arr[idx].ct = current\_time;

arr[idx].tat = arr[idx].ct - arr[idx].at;

arr[idx].wt = arr[idx].tat - arr[idx].bt;

arr[idx].rt = arr[idx].st - arr[idx].at;

arr[idx].is\_completed = 1;

completed++;

}

} else {

current\_time++;

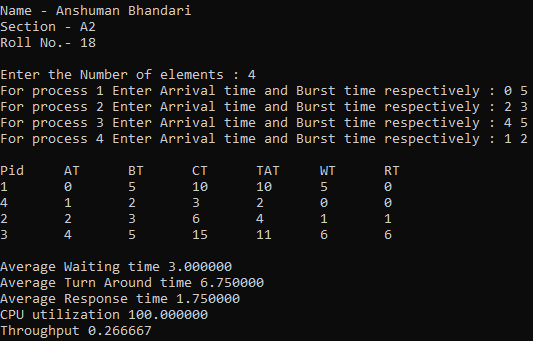
}

}

averageTime(arr, n, total\_it);

}

**OUTPUT:**

****

**Week 4: Process Scheduling**

* + 1. **Priority – process which has highest priority will get CPU first.**

#include <stdio.h>

typedef struct {

int id;

int burst\_time;

int arrival\_time;

int priority;

int completion\_time;

int waiting\_time;

int turnaround\_time;

} Process;

void calculate\_waiting\_and\_turnaround(Process processes[], int n) {

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

processes[i].turnaround\_time = processes[i].completion\_time - processes[i].arrival\_time;

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

}

}

void priority\_scheduling(Process processes[], int n) {

int completed = 0, current\_time = 0;

while (completed < n) {

int highest\_priority = -1;

int selected\_process = -1;

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

if (processes[i].completion\_time == 0 && processes[i].arrival\_time <= current\_time) {

if (selected\_process == -1 || processes[i].priority > highest\_priority) {

highest\_priority = processes[i].priority;

selected\_process = i;

}

}

}

if (selected\_process != -1) {

current\_time += processes[selected\_process].burst\_time;

processes[selected\_process].completion\_time = current\_time;

completed++;

}

else

current\_time++;

}

calculate\_waiting\_and\_turnaround(processes, n);

printf("Gantt Chart: ");

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

printf("P%d ", processes[i].id);

printf("\n");

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

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

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

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

}

printf("Average waiting time: %.2f\n", total\_waiting\_time / n);

printf("Average turnaround time: %.2f\n", total\_turnaround\_time / n);

}

int main() {

printf("\nName - Anshuman Bhandari\nSection - A2\nRoll No.- 18\n\n");

int n;

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

scanf("%d", &n);

Process processes[n];

printf("Enter the burst times of the processes: ");

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

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

processes[i].id = i;

}

printf("Enter the arrival times of the processes: ");

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

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

}

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

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

scanf("%d", &processes[i].priority);

processes[i].completion\_time = 0;

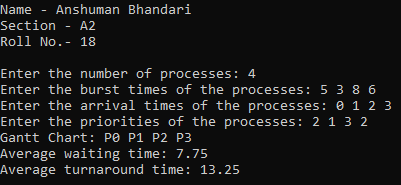
}

priority\_scheduling(processes, n);

return 0;

}

**OUTPUT:**

****

1. **Round Robin – each process is provided a fix time to execute. Once a process is executed for a given time period, it is preempted and other process executes for the given time period.**

#include <stdio.h>

#include <limits.h>

#include <stdlib.h>

typedef struct process {

int pid, at, bt, ct, tat, wt, rt, st, remaining, visited;

} st;

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

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

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

return p1->at > p2->at;

}

int max(int a, int b) {

if (a <= b)

return b;

else

return a;

}

void averageTime(st \*arr, int n, float total\_it) {

float total\_wt = 0, total\_tat = 0, total\_rt = 0;

printf("\nPid\tAT\tBT\tCT\tTAT\tWT\tRT\n");

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

total\_wt += arr[i].wt;

total\_tat += arr[i].tat;

total\_rt += arr[i].rt;

printf("%d\t%d\t%d\t%d\t%d\t%d\t%d\n", arr[i].pid, arr[i].at, arr[i].bt, arr[i].ct, arr[i].tat, arr[i].wt, arr[i].rt);

}

printf("\nAverage Waiting time %f\n", total\_wt / (float)n);

printf("Average Turn Around time %f\n", total\_tat / (float)n);

printf("Average Response time %f\n", total\_rt / (float)n);

printf("CPU utilization %f\n", ((arr[n - 1].ct - total\_it) / (float)arr[n - 1].ct) \* 100);

printf("Throughput %f\n", (float)n / (arr[n - 1].ct - arr[0].at));

}

int main() {

int n, completed = 0, curr\_time = 0, first\_process = 0, qn, front = -1, rear = -1;

int queue[100];

float total\_it = 0;

printf("\nName - Anshuman Bhandari\nSection - A2\nRoll No.- 18\n\n");

printf("Enter the Number of elements : ");

scanf("%d", &n);

st arr[n];

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

printf("For process %d Enter Arrival time and Burst time respectively : ", i + 1);

arr[i].pid = i + 1;

scanf("%d%d", &arr[i].at, &arr[i].bt);

arr[i].remaining = arr[i].bt;

arr[i].visited = 0;

}

printf("Enter time quantam:- ");

scanf("%d", &qn);

qsort(arr, n, sizeof(st), comp);

front = rear = 0;

queue[rear] = 0;

arr[0].visited = 1;

while (completed != n) {

int index = queue[front];

front++;

if (arr[index].bt == arr[index].remaining) {

arr[index].st = max(curr\_time, arr[index].at);

curr\_time = arr[index].st;

if (first\_process == 1)

total\_it += 0;

else

total\_it += arr[index].st - curr\_time;

}

if (arr[index].remaining - qn > 0) {

arr[index].remaining -= qn;

curr\_time += qn;

}

else {

curr\_time += arr[index].remaining;

completed++;

arr[index].ct = curr\_time;

arr[index].tat = arr[index].ct - arr[index].at;

arr[index].wt = arr[index].tat - arr[index].bt;

arr[index].rt = arr[index].st - arr[index].at;

arr[index].remaining = 0;

}

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

if (arr[i].at <= curr\_time && arr[i].visited != 1) {

rear++;

queue[rear] = i;

arr[i].visited = 1;

}

}

if (arr[index].remaining != 0) {

rear++;

queue[rear] = index;

}

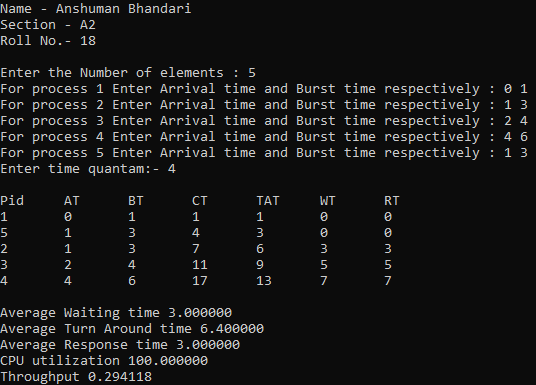
}

averageTime(arr, n, total\_it);

return 0;

}

**OUTPUT:**

****

* 1. **Write a C program to simulate multi-level queue scheduling algorithm considering the following scenario: all the processes in the system are divided into two categories – system processes and user processes. System processes are to be given higher priority than user processes. Use FCFS scheduling for the processes in each queue.**

#include <stdio.h>

typedef struct {

int id,burst\_time,arrival\_time,completion\_time;

int waiting\_time,turnaround\_time,is\_system\_process;

} Process;

void calculate\_waiting\_and\_turnaround(Process processes[], int n) {

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

processes[i].turnaround\_time = processes[i].completion\_time - processes[i].arrival\_time;

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

}

}

void print\_scheduling\_table(Process processes[], int n) {

printf("\nScheduling Table:\n");

printf("PID\tAT\tBT\tCT\tTAT\tWT\tType\n");

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

printf("P%d\t%d\t%d\t%d\t%d\t%d\t%s\n",

processes[i].id,

processes[i].arrival\_time,

processes[i].burst\_time,

processes[i].completion\_time,

processes[i].turnaround\_time,

processes[i].waiting\_time,

processes[i].is\_system\_process ? "System" : "User");

}

}

void multi\_level\_queue\_scheduling(Process processes[], int n) {

int current\_time = 0;

printf("Gantt Chart: ");

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

if (processes[i].is\_system\_process) {

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

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

}

current\_time += processes[i].burst\_time;

processes[i].completion\_time = current\_time;

printf("P%d ", processes[i].id);

}

}

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

if (!processes[i].is\_system\_process) {

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

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

}

current\_time += processes[i].burst\_time;

processes[i].completion\_time = current\_time;

printf("P%d ", processes[i].id);

}

}

printf("\n");

calculate\_waiting\_and\_turnaround(processes, n);

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

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

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

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

}

printf("Average waiting time: %.2f\n", total\_waiting\_time / n);

printf("Average turnaround time: %.2f\n", total\_turnaround\_time / n);

print\_scheduling\_table(processes, n);

}

int main() {

printf("\nName - Anshuman Bhandari\nSection - A2\nRoll No.- 18\n\n");

int n;

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

scanf("%d", &n);

Process processes[n];

printf("Enter the burst times of the processes: ");

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

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

processes[i].id = i;

}

printf("Enter the arrival times of the processes: ");

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

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

printf("Enter the process types (1 for system process, 0 for user process): ");

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

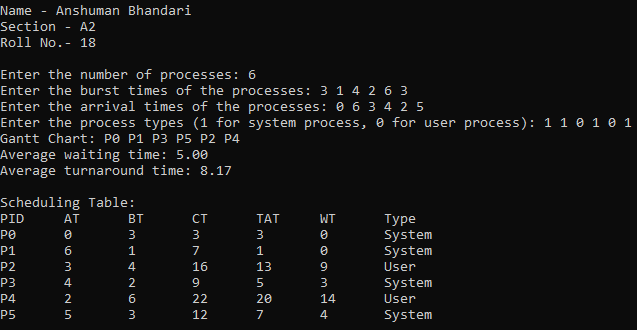
scanf("%d", &processes[i].is\_system\_process);

multi\_level\_queue\_scheduling(processes, n);

return 0;

}

**OUTPUT:**

****

**Week 5: Deadlock**

**I. Write a C program to simulate Bankers algorithm for the purpose of deadlock avoidance.**

#include <stdio.h>

#include <stdbool.h>

typedef struct process {

int max[10], allocated[10], need[10];

} st;

int n, r;

void input(st process[], int available[]) {

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

printf("Enter process[%d] info -\n", i);

printf("Enter Maximum Need : ");

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

scanf("%d", &process[i].max[j]);

}

printf("Enter Allocated Resources for this process : ");

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

scanf("%d", &process[i].allocated[j]);

process[i].need[j] = process[i].max[j] - process[i].allocated[j];

}

}

printf("Enter Available Resources : ");

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

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

}

void showTheInfo(st process[]) {

printf("\nPID\tMaximum\t\tAllocated\tNeed\n");

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

printf("%d\t", i);

for (int j = 0; j < r; j++)

printf("%d ", process[i].max[j]);

printf("\t\t");

for (int j = 0; j < r; j++)

printf("%d ", process[i].allocated[j]);

printf("\t\t");

for (int j = 0; j < r; j++)

printf("%d ", process[i].need[j]);

printf("\n");

}

}

bool applySafetyAlgo(st process[], int available[], int safeSequence[]) {

bool finish[n];

int work[r];

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

work[i] = available[i];

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

finish[i] = false;

bool proceed = true;

int k = 0;

while (proceed) {

proceed = false;

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

bool flag = true;

if (finish[i] == false)

{

for (int j = 0; j < r; j++)

{

if (process[i].need[j] <= work[j]) continue;

else {

flag = false;

break;

}

}

if (flag == false)

continue;

for (int j = 0; j < r; j++)

work[j] = work[j] + process[i].allocated[j];

finish[i] = true;

safeSequence[k++] = i;

proceed = true;

}

}

}

int i;

for (i = 0; i < n && finish[i] == true; i++) continue;

if (i == n) return true;

else return false;

}

bool isSafeState(st process[], int available[], int safeSequence[]) {

if (applySafetyAlgo(process, available, safeSequence) == true)

return true;

return false;

}

int main() {

printf("\nName - Anshuman Bhandari\nSection - A2\nRoll No.- 18\n\n");

printf("Enter No of Process : ");

scanf("%d", &n);

printf("\nEnter No of Resource Instances in system : ");

scanf("%d", &r);

int available[r], safeSequence[n];

st process[n];

input(process, available);

printf("\n");

showTheInfo(process);

printf("\n");

if (isSafeState(process, available, safeSequence)) {

printf("Safe Sequence is : \n");

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

printf("Process no - %d\n", safeSequence[i]);

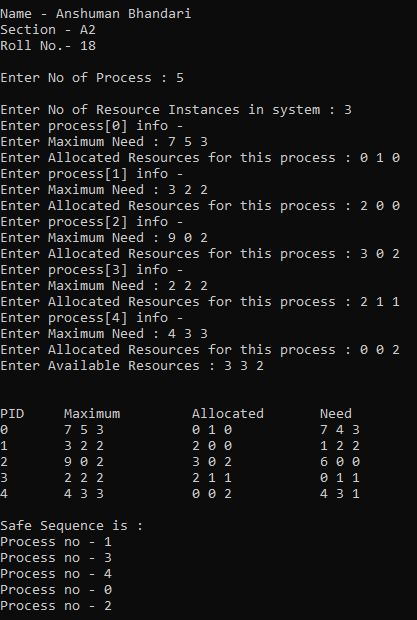
}

else printf("\nSystem is in UNSAFE State\n");

return 0;

}

**OUTPUT:**

****

**II. Write a program to implement deadlock detection algorithm.**

#include <stdio.h>

void calculate\_need(int need[][10], int max[][10], int alloc[][10], int p, int r) {

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

for (int j = 0; j < r; j++)

need[i][j] = max[i][j] - alloc[i][j];

}

int detect\_deadlock(int processes[], int avail[], int max[][10], int alloc[][10], int p, int r) {

int need[10][10];

calculate\_need(need, max, alloc, p, r);

int finish[10] = {0};

int work[10];

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

work[i] = avail[i];

int count = 0;

while (count < p) {

int found = 0;

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

if (finish[i] == 0) {

int j;

for (j = 0; j < r; j++)

if (need[i][j] > work[j]) break;

if (j == r)

{

for (int k = 0; k < r; k++)

work[k] += alloc[i][k];

finish[i] = 1;

found = 1;

count++;

}

}

}

if (found == 0)

{

printf("Deadlock detected\n");

return 1;

}

}

printf("No deadlock detected\n");

return 0;

}

int main() {

printf("\nName - Anshuman Bhandari\nSection - A2\nRoll No.- 18\n\n");

int p, r;

printf("Enter number of processes: ");

scanf("%d", &p);

printf("Enter number of resources: ");

scanf("%d", &r);

int processes[10], avail[10], max[10][10], alloc[10][10];

printf("Enter maximum requirement:\n");

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

for (int j = 0; j < r; j++)

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

printf("Enter allocated matrix:\n");

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

for (int j = 0; j < r; j++)

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

printf("Enter resource vector:\n");

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

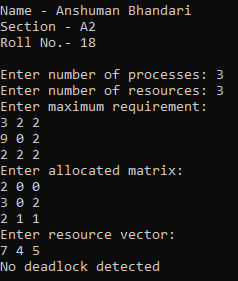
scanf("%d", &avail[i]);

detect\_deadlock(processes, avail, max, alloc, p, r);

return 0;

}

**OUTPUT:**

****

**.**