#odd even

read -p "enter a no.:" n

r=`expr $n % 2`

if [ $r -eq 0 ]

then

echo "$n is even"

else

echo "$n is odd"

fi

#add 2 no.

read -p "enter 1st no.:" a

read -p "enter 2nd no.:" b

sum=$(( $a + $b ))

echo "sum is:" $sum

#factorial of a no.

read -p "enter a no.:" n

fact=1

n1=$n

while [ $n -gt 1 ]

do

fact=$(( $n \* fact ))

n=$(( $n -1 ))

done

echo "factorial $n1:" $fact

#leap year or not

read -p "enter a year:" a

b=`expr $a % 4`

c=`expr $a % 100`

if [[ $b -eq 0 && $c -ne 0 ]]

then

echo "$a is leap year"

else

echo "$a isn't leap year"

fi

#convert °c into F

read -p "enter the temp in °c:" c

f=$(echo "scale=2;$c \* 9/5 + 32" | bc)

echo "$c °c -> $f F"

#swapp 2 no.

read -p "enter 1st no.:" a

read -p "enter 2nd no.:" b

a1=$a

b1=$b

temp=$a

a=$b

b=$temp

echo "after swapping:$a1 $b1 -> $a $b"

#prime or not

read -p "enter a no.:" n

for (( i=2; i<=$(( $n / 2 )); i++))

do

if [[ $(( $n % $i )) -eq 0 ]]

then

echo "$n isn't prime no."

exit

fi

done

echo "$n is prime no."

#prime in range

read -p "enter the upper limit:" n

echo "prime numbers between 1 to $n"

for (( i=1; i<=$n; i++ ))

do

flag=0

for(( j=2; j\*j<$(( $i + 1 )); j++ ))

do

if [[ $(( $i % $j )) -eq 0 ]]

then

flag=1

break

fi

done

if [[ flag -eq 0 ]]

then

echo "$i-prime no."

fi

done

//process creation & parent child relation(fork() system call)

#include<stdio.h>

#include<unistd.h>

int main(){

int f\_val=fork();

if(f\_val==0){

printf("from child, pid:%d, ppid:%d, fork return val:%d\n",getpid(),getppid(),f\_val);

}

else if(f\_val>0){

printf("from parent, pid:%d, ppid:%d, fork return val:%d\n",getpid(),getppid(),f\_val);

}

else{

printf("error in fork\n");

}

return 0;

}

//fork() 2

#include<stdio.h>

#include<unistd.h>

int main(){

int f\_val=fork();

if(f\_val==0){

printf("this line is from child\n");

}

else{

printf("this line is from parent, fork return val:%d\n",f\_val);

}

return 0;

}

//getpid()

#include<stdio.h>

#include<unistd.h>

int main(){

printf("this line is from parent, ppid:%d\n",getpid()); //ppid=process identifier of parent process

int f\_val=fork();

if(f\_val==0){

printf("after fork() system call,this line is from child, pid:%d\n",getpid()); //pid=process identifier

}

else{

printf("after fork() system call,this line is from cparent, ppid:%d, fork return val:%d\n",getppid(),f\_val);

}

return 0;

}

//distinguish between parent & child process

#include<stdio.h>

#include<unistd.h>

#include<sys/types.h>

#include<string.h>

#define MAX\_COUNT 5

#define BUF\_SIZE 100

void main(){

int i;

pid\_t pid;

char buf[BUF\_SIZE];

fork();

pid=getpid();

for(i=1;i<=MAX\_COUNT;i++){

sprintf(buf,"this line is from pid:%d, value:%d\n",pid,i);

write(1,buf,strlen(buf));

}

}

//distinguish parent from child

#include<stdio.h>

#include<unistd.h>

#define MAX\_COUNT 5

void child\_process();

void parent\_process();

void main(){

int f\_val=fork();

if(f\_val==0){

child\_process();

}

else{

parent\_process();

}

}

void child\_process(){

for(int i=1;i<=MAX\_COUNT;i++){

printf("this line is from child, value:%d\n",i);

}

printf("\*\*child process ends\*\*\n");

}

void parent\_process(){

for(int i=1;i<=MAX\_COUNT;i++){

printf("this line is from parent, value:%d\n",i);

}

printf("\*\*parent process ends\*\*\n");

}

//combining fork() & exec()

#include<stdio.h>

#include<unistd.h>

#include<stdlib.h>

#include<sys/types.h>

#include<sys/wait.h>

int main(){

char \*argv[3]={"command line",".",NULL};

int f\_val=fork();

if(f\_val==0){ //child process

execvp("find",argv);

}

//put parent in sleep,let the child finish its execution

wait((int \*)2);

printf("parent finished its execution,we won't get child here, we can only see it for once\n");

return 0;

}

//fcfs

#include<stdio.h>

#include<stdlib.h>

typedef struct{

int pid,bt,wt,tt;

}sp;

int main(){

int i,n,towt=0,tott=0,tbm=0;

sp \*p;

printf("FCFS scheduling..\n");

printf("enter the no. of processes: ");

scanf("%d",&n);

p=(sp \*)malloc(n\*sizeof(sp));

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

p[i].pid=i+1;

printf("\nenter the bt of process id %d: ",p[i].pid);

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

}

//print & cal

printf("\nprocess scheduling..\n");

printf("process\tburst\twaiting\tturnaround\n");

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

tbm+=p[i].bt;

p[i].tt=tbm;

p[i].wt=tbm-p[i].bt;

towt+=p[i].wt;

tott+=p[i].tt;

printf("%d\t%d\t%d\t%d\n",p[i].pid,p[i].bt,p[i].wt,p[i].tt);

}

printf("avg wt:%.2f\n",(float)towt/n);

printf("avg tt:%.2f\n",(float)tott/n);

free(p);

return 0;

}

//fcfs with at

#include<stdio.h>

#include<stdlib.h>

typedef struct{

int pid,bt,wt,tt,at;

}sp;

int main(){

int i,j,n,towt=0,tott=0,tbm=0;

sp \*p,t;

printf("FCFS with at scheduling..\n");

printf("enter the no. of processes: ");

scanf("%d",&n);

p=(sp \*)malloc(n\*sizeof(sp));

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

p[i].pid=i+1;

printf("\nenter the bt of process id %d: ",p[i].pid);

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

printf("enter the at of process id %d: ",p[i].pid);

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

}

//sort processes acc to their at

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

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

if(p[i].at>p[j].at){

t=p[i];

p[i]=p[j];

p[j]=t;

}

}

}

//print & cal

printf("\nprocess scheduling..\n");

printf("process\tburst\tarrival\twaiting\tturnaround\n");

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

tbm+=p[i].bt;

p[i].tt=tbm-p[i].at;

p[i].wt=tbm-p[i].bt-p[i].at;

towt+=p[i].wt;

tott+=p[i].tt;

printf("%d\t%d\t%d\t%d\t%d\n",p[i].pid,p[i].bt,p[i].at,p[i].wt,p[i].tt);

}

printf("avg wt:%.2f\n",(float)towt/n);

printf("avg tt:%.2f\n",(float)tott/n);

free(p);

return 0;

}

//sjf(non preemptive)

#include<stdio.h>

#include<stdlib.h>

typedef struct{

int pid,bt,wt,tt;

}sp;

int main(){

int i,j,n,towt=0,tott=0,tbm=0;

sp \*p,t;

printf("SJF scheduling..\n");

printf("enter the no. of processes: ");

scanf("%d",&n);

p=(sp \*)malloc(n\*sizeof(sp));

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

p[i].pid=i+1;

printf("\nenter the bt of process id %d: ",p[i].pid);

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

}

//sort processes acc to their bt

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

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

if(p[i].bt>p[j].bt){

t=p[i];

p[i]=p[j];

p[j]=t;

}

}

}

//print & calc

printf("\nscheduling...\n");

printf("process\tburst\twaiting\tturnaround\n");

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

tbm+=p[i].bt;

p[i].tt=tbm;

p[i].wt=tbm-p[i].bt;

towt+=p[i].wt;

tott+=p[i].tt;

printf("%d\t%d\t%d\t%d\n",p[i].pid,p[i].bt,p[i].wt,p[i].tt);

}

printf("avg wt:%.2f\n",(float)towt/n);

printf("avg tt:%.2f\n",(float)tott/n);

free(p);

return 0;

}

//Preemptive SJF scheduling

#include<stdio.h>

#include<stdlib.h>

#include <limits.h>

typedef struct {

int pid, bt, wt, tt, at, remaining\_time;

} sp;

int main() {

int i, j, n, tbm = 0, towt = 0, tott = 0;

printf("Preemptive SJF scheduling with arrival time\n");

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

scanf("%d", &n);

sp \*p, t;

p = (sp \*)malloc(n \* sizeof(sp));

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

p[i].pid = i + 1;

printf("\nEnter burst time of process id %d: ", p[i].pid);

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

printf("Enter arrival time of process id %d: ", p[i].pid);

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

p[i].remaining\_time = p[i].bt;

}

// Sort processes by their arrival time

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

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

if (p[i].at > p[j].at) {

t = p[i];

p[i] = p[j];

p[j] = t;

}

}

}

printf("\nProcess scheduling\n");

printf("Process\tBurst\tArrival\tWaiting\tTurnaround\n");

int current\_time = 0;

while (1) {

int idx = -1, min\_bt = INT\_MAX;

// Find the process with the shortest remaining burst time at the current time

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

if (p[i].at <= current\_time && p[i].remaining\_time > 0 && p[i].remaining\_time < min\_bt) {

min\_bt = p[i].remaining\_time;

idx = i;

}

}

if (idx == -1) {

// No processes remaining

break;

}

// Update current time

current\_time++;

// Update waiting and turnaround times

if (p[idx].remaining\_time == p[idx].bt) {

// First time the process is executed

p[idx].wt = current\_time - p[idx].at - 1;

}

// Update remaining time

p[idx].remaining\_time--;

// Check if the process is completed

if (p[idx].remaining\_time == 0) {

p[idx].tt = current\_time - p[idx].at; //when p[idx].remaining\_time,then current\_time==p[i].bt(idx==i)

tbm += p[idx].bt;

towt += p[idx].wt;

tott += p[idx].tt;

printf("%d\t\t%d\t%d\t%d\t%d\n", p[idx].pid, p[idx].bt, p[idx].at, p[idx].wt, p[idx].tt);

}

}

printf("\nAverage waiting time: %.2f\n", (float)towt / n);

printf("Average turnaround time: %.2f\n", (float)tott / n);

free(p);

}

//rr

#include <stdio.h>

int main() {

int n;

printf("Enter Total Number of Processes: ");

scanf("%d", &n);

int wt = 0, tt = 0;

int burst[n], rburst[n];

int q;

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

printf("Enter Burst Time for Process %d: ", i + 1);

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

rburst[i] = burst[i];

}

printf("Enter Time Slice (Quantum): ");

scanf("%d", &q);

int total\_time = 0;

int completed\_processes = 0;

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

int curp = 0;

while (completed\_processes < n) {

if (rburst[curp] > 0) {

int exet;

if (rburst[curp] > q) {

exet = q;

} else {

exet = rburst[curp];

}

total\_time += exet;

rburst[curp] -= exet;

if (rburst[curp] == 0) {

completed\_processes++;

int turnaround\_time = total\_time;

int waiting\_time = turnaround\_time - burst[curp];

printf("%d\t\t%d\t\t%d\t\t%d\n", curp + 1, burst[curp], turnaround\_time, waiting\_time);

wt += waiting\_time;

tt += turnaround\_time;

}

curp = (curp + 1) % n;

} else {

curp = (curp + 1) % n;

}

}

float avg\_waiting\_time = (float)wt / n;

float avg\_turnaround\_time = (float)tt / n;

printf("\nAverage Waiting Time: %f", avg\_waiting\_time);

printf("\nAverage Turnaround Time: %f\n", avg\_turnaround\_time);

}

//rr with at

#include <stdio.h>

int main() {

int n;

printf("Enter Total Number of Processes: ");

scanf("%d", &n);

int wt = 0, tt = 0;

int arrt[n],burst[n], rburst[n];

int q;

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

printf("\nEnter Burst Time for Process %d: ", i + 1);

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

printf("\nEnter arrival Time for Process %d: ", i + 1);

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

rburst[i] = burst[i];

}

printf("Enter Time Slice (Quantum): ");

scanf("%d", &q);

int total\_time = 0;

int completed\_processes = 0;

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

int curp = 0;

while (completed\_processes < n) {

if (rburst[curp] > 0) {

int exet;

if (rburst[curp] > q) {

exet = q;

} else {

exet = rburst[curp];

}

total\_time += exet;

rburst[curp] -= exet;

if (rburst[curp] == 0) {

completed\_processes++;

int turnaround\_time = total\_time-arrt[curp];

int waiting\_time = turnaround\_time - burst[curp];

printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\n", curp + 1, burst[curp], arrt[curp],turnaround\_time,

waiting\_time);

wt += waiting\_time;

tt += turnaround\_time;

}

curp = (curp + 1) % n;

} else {

curp = (curp + 1) % n;

}

}

float avg\_waiting\_time = (float)wt / n;

float avg\_turnaround\_time = (float)tt / n;

printf("\nAverage Waiting Time: %f", avg\_waiting\_time);

printf("\nAverage Turnaround Time: %f\n", avg\_turnaround\_time);

}