//odd even

```
echo "Enter the Number"
read n
r='expr $n % 2'
if [ $r -eq 0 ]
then
   echo "$n is Even number"
else
   echo "$n is Odd number"
fi
```

//factorial

```
echo "Enter a number"
read num
fact=1
while [ $num -gt 1 ]
do
fact=`expr $fact \* $num`
num=`expr $num - 1`
done
echo $fact
```

//switch

```
echo "ENTER 1st NUMBER:"
read a
echo "ENTER 2nd NUMBER:"
```

```
read b
i=1
while [ $i -eq 1 ]; do
   echo "!!!MENU!!!"
   echo "1.ADDITION OF $a AND $b"
   echo "2.SUBTRACTION OF $a AND $b"
   echo "3.MULTIPLICATION OF $a AND $b"
   echo "4.DIVISION OF $a AND $b"
   echo "5.EXPONENTIAL FUNCTION $a^$b"
   echo "6.EXIT"
   echo "ENTER YOUR CHOICE:"
   read choice
   case $choice in
      1)
         sum=`expr $a + $b`
         echo "SUM: $sum"
         ;;
      2)
         diff=`expr $a - $b`
         echo "DIFFERENCE: $diff"
          ;;
      3)
         pro=`expr $a \* $b`
         echo "PRODUCT: $pro"
         ;;
      4)
         echo "DIVISION: $(echo "scale=2; $a/$b" | bc)"
         ;;
      5)
         exp=1
         for (( i=1 ; i <= b ; i++ )); do
            exp=`expr $a \* $exp`
```

```
        done
        ;;
    6)
        break
        ;;
    *)
        echo "WRONG ENTRY"
        ;;
  esac
  echo "$a^$b = $exp"
done
echo "PRESS 1 TO CONTINUE:"
read i
```

//process management fork

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
void forkexample()
{
        pid_t p;
        p = fork();
        if(p<0)
        {
        perror("fork fail");
```

```
        exit(1);
        }
        //child process because return value zero
        else if(p==0){
                printf("Hello from Child!\n");
        }
        //parent process because return value non-zero.
        else{
                printf("Hello from Parent\n");
        }
}
int main(){
        forkexample();
        return 0;
}
```

```
//process management of exec system call

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

int main()

{

//A null terminated array of character
```

```
//pointers

char *args[]={"./EXEC",NULL};
execv(args[0],args);

//process(execDemo.c) is replaced by another process (EXEC.c)

printf("Ending....");

return 0;
}



//inretprocess communication related process using pipes

#include<stdio.h>
#include<unistd.h>
int main(){
        int pipefds[2];
        int returnstatus;
        int pid;
        char writemessages[2][20]={"KCE", "Karpagam"};
        char readmessage[20];
        returnstatus = pipe(pipefds);
        if(returnstatus == -1){
                printf("Unable to create pipe\n");
                return 1;
        }
        pid = fork();
        // Child process
        if (pid == 0) {
```

```
        read(pipefds[0], readmessage, sizeof(readmessage));
        printf("Child Process - Reading from pipe Message 1 is %s\n", readmessage);
        read(pipefds[0], readmessage, sizeof(readmessage));
        printf("Child Process - Reading from pipe Message 2 is %s\n", readmessage);
        }
        else { //Parent process
                printf("Parent Process - Writing to pipe Message 1 is %s\n", writemessages[0]);
                write(pipefds[1], writemessages[0], sizeof(writemessages[0]));
                printf("Parent Process - Writing to pipe Message 2 is %s\n", writemessages[1]);
                write(pipefds[1], writemessages[1], sizeof(writemessages[1]));
        }
        return 0;
}
```

```
//interprocess communication among unrelated process using messagte queue

#include <stdio.h>
#include <sys/msg.h>

#define MAX 10

//structure for message qurue
struct mesg_buffer{
long mesg_type;
char mesg_text[100];
}message;
int main()
{
        key_t key;
```

```
        int msgid;

//flok to generate unique key
key=ftok("progfile", 65);

//msgget creates a message queue
//and returns identifier
msgid=msgget(key, 0666 | IPC_CREAT);
message.mesg_type = 1;

printf("Write Data: ");

fgets(message.mesg_text,MAX,stdin);

//magand to send message
msgsnd(msgid, &message, sizeof(message), 0);

//display the message
printf("Data send is: %s\n", message.mesg_text);

return 0;
}



// C Program for Message Queue (Reader Process)
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/msg.h>

//structure for message queue
struct mesg_buffer{
long mesg_type;
```

```
char mesg_text[100];
} message;

int main()
{
key_t key;
int msgid;

//flok to generate unique key
key=ftok(" progfile", 65);
//migget creates a message queue and returns identifier
msgid=msgget(key, 0666 | IPC_CREAT);

//msgrev to receive message
msgrcv(msgid, &message, sizeof(message), 1, 0);


//display the message
printf("Data Received is: %s\n",message.mesg_text);

//to destroy the message queue
msgctl(msgid, IPC_RMID, NULL);

return 0;

}



// cpu sheudeling algorithm  (sjf)
```

```
#include <stdio.h>
int main()
{
// Matrix for storing Process Id, Burst
// Time, Average Waiting Time & Average
// Turn Around Time.
int A[100][4];
int i, j, n, total = 0, index, temp;
float avg_wt, avg_tat;
printf("Enter number of process: ");
scanf("%d", &n);
printf("Enter Burst Time:\n");
// User Input Burst Time and alloting Process Id.
for (i = 0; i < n; i++) {
printf("P%d: ", i + 1);
scanf("%d", &A[i][1]);
A[i][0] = i + 1;

}
// Sorting process according to their Burst Time.
for (i = 0; i < n; i++) {
index = i;
for (j = i + 1; j < n; j++)
if (A[j][1] < A[index][1])
index = j;
temp = A[i][1];
A[i][1] = A[index][1];
A[index][1] = temp;
temp = A[i][0];
A[i][0] = A[index][0];
A[index][0] = temp;
}
```

```
A[0][2] = 0;
// Calculation of Waiting Times
for (i = 1; i < n; i++) {
A[i][2] = 0;
for (j = 0; j < i; j++)
A[i][2] += A[j][1];
total += A[i][2];
}
avg_wt = (float)total / n;
total = 0;
printf("P BT WT TAT\n");
// Calculation of Turn Around Time and printing the
// data.
for (i = 0; i < n; i++) {
A[i][3] = A[i][1] + A[i][2];
total += A[i][3];
printf("P%d %d %d %d\n", A[i][0],
A[i][1], A[i][2], A[i][3]);

}
avg_tat = (float)total / n;
printf("Average Waiting Time= %f", avg_wt);
printf("\nAverage Turnaround Time= %f", avg_tat);
}
```

```
//fcfs

#include <stdio.h>
// Function to find the waiting time for all processes
int waitingtime(int proc[], int n,
```

```
int burst_time[], int wait_time[]) {
// waiting time for first process is 0
wait_time[0] = 0;
int i;
// calculating waiting time
for ( i = 1; i < n ; i++ )
wait_time[i] = burst_time[i-1] + wait_time[i-1] ;
return 0;
}
// Function to calculate turn around time
int turnaroundtime( int proc[], int n,
int burst_time[], int wait_time[], int tat[]) {
// calculating turnaround time by adding
// burst_time[i] + wait_time[i]
int i;
for ( i = 0; i < n ; i++)
tat[i] = burst_time[i] + wait_time[i];
return 0;
}
//Function to calculate average time
int avgtime( int proc[], int n, int burst_time[]) {
int wait_time[n], tat[n], total_wt = 0, total_tat = 0;
int i;
//Function to find waiting time of all processes
waitingtime(proc, n, burst_time, wait_time);
//Function to find turn around time for all processes
turnaroundtime(proc, n, burst_time, wait_time, tat);
//Display processes along with all details
printf("Processes  Burst   Waiting  Turn around\n");
// Calculate total waiting time and total turn
// around time
for ( i=0; i<n; i++) {
```

```
total_wt = total_wt + wait_time[i];
total_tat = total_tat + tat[i];
printf("\t%d\t%d\t%d\t%d\n", i+1, burst_time[i], wait_time[i], tat[i]);
}
printf("\nAverage waiting time = %f", (float)total_wt / (float)n);
printf("\nAverage turn around time = %f", (float)total_tat / (float)n);
return 0;
}
// main function
int main() {
//process id's
int proc[] = { 1, 2, 3};
int n = sizeof proc / sizeof proc[0];
//Burst time of all processes
int burst_time[] = {5, 8, 12};
avgtime(proc, n, burst_time);
return 0;
}
```

```
//priority schudulinh

#include <stdio.h>

void calculate_waiting_time(int n, int burst_time[], int waiting_time[], int priori_timr[])
{
    waiting_time[0] = 0;

    // Calculate waiting time for each process

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

```
    waiting_time[i] = burst_time[i - 1] + waiting_time[i - 1];
    }

}

void calculate_turnaround_time(int n, int burst_time[], int waiting_time[], int turnaround_time[]) {

// Calculate turnaround time for each process

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

    turnaround_time[i] = burst_time[i] + waiting_time[i];
    }
}
void calculate_average_times(int n, int waiting_time[], int turnaround_time[]) { int
total_waiting_time = 0, total_turnaround_time = 0;

printf("Process\t Waiting Time\t Turnaround Time\n");

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

printf("P%d\t\t%d\t\t%d\n", i + 1, waiting_time[i], turnaround_time[i]);
total_waiting_time += waiting_time[i];
total_turnaround_time += turnaround_time[i];
}
double average_waiting_time = (double)total_waiting_time/n;
double average_turnaround_time = (double)total_turnaround_time / n;

printf("\nAverage Waiting Time: %.21f\n", average_waiting_time);
printf("Average Turnaround Time: %.21f\n", average_turnaround_time);

}
```

```
void priority_scheduling(int n, int burst_time[], int priority[]) {

int waiting_time[n], turnaround_time[n];

calculate_waiting_time(n, burst_time, waiting_time, priority);

calculate_turnaround_time(n, burst_time, waiting_time, turnaround_time);

calculate_average_times(n, waiting_time, turnaround_time);

}

int main() {

// Input: List of processes with their burst time and priority

int n = 4;

int burst_time[] = {6, 8, 7, 3};

int priority[] = {2, 1, 4, 3};

priority_scheduling(n, burst_time, priority);

return 0;
}
```

```
//roundrobin
```

```
#include<stdio.h>
void main()
{
// initlialize the variable name
int i, NOP, sum=0,count=0, y, quant, wt=0, tat=0, at[10], bt[10], temp[10];
float avg_wt, avg_tat;
printf(" Total number of process in the system: ");
scanf("%d", &NOP);
y = NOP; // Assign the number of process to variable y
// Use for loop to enter the details of the process like Arrival time and the Burst Time
for(i=0; i<NOP; i++)
{
printf("\n Enter the Arrival and Burst time of the Process[%d]\n", i+1);
printf(" Arrival time is: \t"); // Accept arrival time
scanf("%d", &at[i]);
printf(" \nBurst time is: \t"); // Accept the Burst time
scanf("%d", &bt[i]);
temp[i] = bt[i]; // store the burst time in temp array
}
// Accept the Time qunat
printf("Enter the Time Quantum for the process: \t");
scanf("%d", &quant);
// Display the process No, burst time, Turn Around Time and the waiting time
printf("\n Process No \t\t Burst Time \t\t TAT \t\t Waiting Time ");
for(sum=0, i = 0; y!=0; )
{
if(temp[i] <= quant && temp[i] > 0) // define the conditions
{
sum = sum + temp[i];
temp[i] = 0;
count=1;
```

```
}
else if(temp[i] > 0)
{
temp[i] = temp[i] - quant;
sum = sum + quant;
}
if(temp[i]==0 && count==1)
{
y--; //decrement the process no.
printf("\nProcess No[%d] \t\t %d\t\t\t\t %d\t\t\t %d", i+1, bt[i], sum-at[i], sum-at[i]-bt[i]);
wt = wt+sum-at[i]-bt[i];
tat = tat+sum-at[i];
count =0;
}
if(i==NOP-1)
{
i=0;
}
else if(at[i+1]<=sum)
{
i++;
}
else
{
i=0;
}
}
// represents the average waiting time and Turn Around time
avg_wt = wt * 1.0/NOP;
avg_tat = tat * 1.0/NOP;
printf("\n Average Turn Around Time: \t%f", avg_wt);
printf("\n Average Waiting Time: \t%f", avg_tat);
```

```
getch();
}


//mplementation of fist fit


#include <stdio.h>
#define MEMORY_SIZE 1000
// Structure to represent a memory block
struct MemoryBlock {
int start_address;
int size;
int allocated;
};
// Function to initialize the memory blocks
void initialize_memory(struct MemoryBlock memory[], int n) {
      int i;
for ( i = 0; i < n; i++) {
memory[i].start_address = -1; // Indicates memory block is free
memory[i].size = 0;
memory[i].allocated = 0;
}
}
// Function to allocate memory using First Fit algorithm
void first_fit_allocation(struct MemoryBlock memory[], int n, int process_id, int size) {
      int i;
for ( i = 0; i < n; i++) {
if (memory[i].allocated == 0 && memory[i].size >= size) {
// Allocate memory
memory[i].allocated = 1;
printf("Process P%d allocated at memory block %d, size %d\n", process_id,
```

```
memory[i].start_address, size);
memory[i].size -= size;
return;
}
}
printf("Process P%d cannot be allocated due to insufficient memory\n", process_id);
}
// Function to deallocate memory
void deallocate_memory(struct MemoryBlock memory[], int n, int process_id) {
    int i;
for (i = 0; i < n; i++) {
if (memory[i].allocated == 1 && memory[i].start_address == process_id) {
memory[i].allocated = 0; // Free memory block
printf("Memory block allocated to Process P%d is deallocated\n", process_id);
return;
}
}
printf("Process P%d not found or already deallocated\n", process_id);
}
// Function to display memory status
void display_memory_status(struct MemoryBlock memory[], int n) {
printf("\nMemory Status:\n");
printf("Memory Block\t\tAllocated\tSize\n");
int i;
for (i = 0; i < n; i++) {
printf("%d\t\t\t", i);
if (memory[i].allocated == 1) {
printf("Yes\t\t%d\n", memory[i].size);
} else {
printf("No\t\t%d\n", memory[i].size);
}
}
```

```
}
int main() {
// Initialize memory blocks
int n = 5; // Number of memory blocks
struct MemoryBlock memory[n];
initialize_memory(memory, n);
// Allocate memory to processes
first_fit_allocation(memory, n, 1, 200);
first_fit_allocation(memory, n, 2, 300);
first_fit_allocation(memory, n, 3, 150);
first_fit_allocation(memory, n, 4, 400);
// Display memory status after allocation
display_memory_status(memory, n);
// Deallocate memory
deallocate_memory(memory, n, 2);
// Display memory status after deallocation
display_memory_status(memory, n);
return 0;
}
```

```
//LUR page replacement

#include<stdio.h>
#include<stdbool.h>
#define MAX_FRAMES 3
bool is_Page_in_memory(int page,int frames[],int n){
for(int i=0;i<n;i++){
if(frames[i]==page){
return true;
```

```
}
}
return false;
}
int find_lru_index(int page_order[],int n,int frame_count){
int lru_index=-1;
int farthest_index=-1;
for(int i=0;i<frame_count;i++){
int j;
for(j=n-1;j>0;j--){
if(frames[i]==page_orders[j]){
if(j>farthest_index){
farthest_index=j;
lru_index=i;
}
break;
}
}
if(j==0){
return i;
}
}
return lru_index;
}
void lru_page_replacement(int reference_string[],int reference_length,int frames[],int frame_count){
int page_order[reference_length];
int page_faults=0;
for(int i=0;i<reference_length;i++){
int page=reference_string[i];
page_order[i]=page;
if(!is_page_in_memory(page,frames,frame_count)){
page_faults++;
```

```
int lru_index=find_lru_index(page_order,i,frames,frame_count);
frames[lru_index]=page;
}
printf("Reference:%d,Frames:",page);
for(int j=0;j<frame_count;j++){
printf("%d",frames[j]);
}
printf("\n");
}
printf("\nTotal Page Faults:%d\n",page_faults);
}
int main(){
int reference_string[]={1,2,3,4,1,2,5,1,2,3,4,5};
int reference_length=sizeof(reference_string)/sizeof(reference_string[0]);
int frames[MAX_FRAMES]={-1};
int frame_count=sizeof(frames)/sizeof(frames[0]);
printf("LRU Page Replacement Stimulation:\n");
lru_page_replacement(reference_string,reference_length,frames,frame_count);
return 0;
}
```

fifo

```
#include <stdio.h>
#include <stdbool.h>
#define MAX_FRAMES 3 // Maximum number of frames in memory
// Function to check if a page exists in memory
bool is_page_in_memory(int page, int frames[], int frame_count) {
    int i;
```

```
for ( i = 0; i < frame_count; i++) {
if (frames[i] == page) {
return true;
}
}
return false;
}
// Function to simulate page replacement using FIFO algorithm
void fifo_page_replacement(int reference_string[], int reference_length, int frames[], int
frame_count) {
int page_faults = 0,i,j;
int oldest_page_index = 0;
for (i = 0; i < reference_length; i++) {
int page = reference_string[i];
if (!is_page_in_memory(page, frames, frame_count)) {
page_faults++;
frames[oldest_page_index] = page;
oldest_page_index = (oldest_page_index + 1) % frame_count;
}
printf("Reference: %d, Frames: ", page);
for (j = 0; j < frame_count; j++) {
printf("%d ", frames[j]);
}
printf("\n");
}
printf("\nTotal Page Faults: %d\n", page_faults);
}
int main() {
int reference_string[] = {1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5}; // Reference string
int reference_length = sizeof(reference_string) / sizeof(reference_string[0]);
int frames[MAX_FRAMES] = {-1}; // Initialize frames with -1 indicating empty frame
int frame_count = sizeof(frames) / sizeof(frames[0]);
```

```
printf("FIFO Page Replacement Simulation:\n");
fifo_page_replacement(reference_string, reference_length, frames, frame_count);
return 0;
}
```