OS Lab Evaluation Codes

CPU Scheduling (FCFS)

```
#include <stdio.h>

void swap(int* a, int* b){
   int temp = *a;
   *a = *b;
   *b = temp;
}

void sort(int p[], int at[], int bt[], int size){
   for(int i=0; i<size; i++){
      for(int j=0; j<size-i-1; j++){
         if(at[j]>at[j+1]){
            swap(&at[j], &at[j+1]);
            swap(&bt[j], &bt[j+1]);
            swap(&p[j], &p[j+1]);
         }
      }
   }
}

double sum(int arr[], int size){
   double sum=0;
   for(int k=0; k<size; k++)
      sum+=arr[k];
   return sum;
}

int main(){
   int N;
   printf("Enter no of processes: ");
   scanf("%d", &N);

   //input
   int at[N], bt[N], p[N], wait[N], tat[N], ct[N], idle_time=0;
   for(int i=0; i<N; i++){
      printf("Enter arrival and burst time for process %d\n", i+1);
      scanf("%d\n%d", &at[i], &bt[i]);
      p[i] = i+1;
   }

   sort(p, at, bt, N);

   //fcfs
   wait[0] = 0;
   ct[0] = at[0] + bt[0];

   for(int i=0; i<N; i++){
      idle_time=0;
      wait[i] = ct[i] - bt[i] - at[i];
```

```
    if(at[i+1] > ct[i]) idle_time += at[i+1] - ct[i];
    ct[i+1] = ct[i] + bt[i+1] + idle_time;
    tat[i] = wait[i] + bt[i];
  }

  //display
  printf("\nProcess\tArrival\tBurst\tWait\tTAT\n");
  for(int i=0; i<N; i++)
    printf("%d\t%d\t%d\t%d\t%d\n", p[i], at[i], bt[i], wait[i], tat[i]);

  printf("\nAvg wait time: %f", sum(wait, N)/N);
  printf("\nAvg turn around time: %f\n", sum(tat, N)/N);
}
```

CPU Scheduling (SJF)

```
#include <stdio.h>
#include <limits.h>
#include <stdbool.h>

struct process
{
  int pid;
  int at;
  int st; // start time
  int bt;
  int rt;
  int ct;
  int wt;
  int tat;
};

int main()
{
  int N;
  printf("Enter no of processes: ");
  scanf("%d", &N);

  struct process P[N];
  int curr_time = 0, idle_time = 0, completed = 0;
  int is_completed[100] = {false};

  for (int i = 0; i < N; i++){
    printf("Enter arrival time and burst time for process %d:\n", i+1);
    scanf("%d %d", &P[i].at, &P[i].bt);
    P[i].rt = P[i].bt;
    P[i].pid = i + 1;
  }

  while (completed != N){
    int min_proc = -1; // returns the index for the proc w min bt
    int min_bt = INT_MAX; // largest value int can hold
    for(int i = 0; i < N; i++){
```

```
            if (P[i].at <= curr_time && is_completed[i] == false){
                if (P[i].rt < min_bt){
                    min_bt = P[i].rt;
                    min_proc = i;
                }

                if (P[i].rt == min_bt){
                    if (P[i].at < P[min_proc].at){
                        min_bt = P[i].rt;
                        min_proc = i;
                    }
                }
            }
        }

        if (min_proc == -1){
            // no proc found w min CPU bt in ready queue till curr_time
            curr_time++;
            idle_time++;
        }

        else{
            if (P[min_proc].rt == P[min_proc].bt){
                P[min_proc].st = curr_time;
            }
            P[min_proc].rt--;
            curr_time++;

            if (P[min_proc].rt == 0){
                P[min_proc].ct = curr_time;
                P[min_proc].tat = P[min_proc].ct - P[min_proc].at;
                P[min_proc].wt = P[min_proc].tat - P[min_proc].bt;
                completed++;
                is_completed[min_proc] = true;
            }
        }
    }

    printf("\nPId\tArr\tBT\tWait\tTAT\n");
    for (int i = 0; i < N; i++)
        printf("%d\t%d\t%d\t%d\t%d\n", P[i].pid, P[i].at, P[i].bt, P[i].wt, P[i].tat);

    int sum_wt = 0, sum_tat = 0;
    for (int i = 0; i < N; i++){
        sum_wt += P[i].wt;
        sum_tat += P[i].tat;
    }
    printf("\nAvg waiting time: %f", (float)sum_wt / N);
    printf("\nAvg turn around time: %f", (float)sum_tat / N);
    printf("\nIdle time: %d\n", idle_time);
}
```

CPU Scheduling (Round Robin)

```
#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);

}
```

CPU Scheduling (Priority)

```
#include <stdio.h>

//Function to swap two variables
void swap(int *a,int *b)
{
    int temp=*a;
    *a=*b;
    *b=temp;
}
int main()
{
    int n;
    printf("Enter Number of Processes: ");
    scanf("%d",&n);

    // b is array for burst time, p for priority and index for process id
    int b[n],p[n],index[n];
    for(int i=0;i<n;i++)
    {
        printf("Enter Burst Time and Priority Value for Process %d: ",i+1);
        scanf("%d %d",&b[i],&p[i]);
        index[i]=i+1;
    }
    for(int i=0;i<n;i++)
    {
        int a=p[i],m=i;

        //Finding out highest priority element and placing it at its desired position
        for(int j=i;j<n;j++)
        {
            if(p[j] > a)
            {
                a=p[j];
```

```
            m=j;
          }
        }

        //Swapping processes
        swap(&p[i], &p[m]);
        swap(&b[i], &b[m]);
        swap(&index[i],&index[m]);
    }

    // T stores the starting time of process
    int t=0;

    //Printing scheduled process
    printf("Order of process Execution is\n");
    for(int i=0;i<n;i++)
    {
        printf("P%d is executed from %d to %d\n",index[i],t,t+b[i]);
        t+=b[i];
    }
    printf("\n");
    printf("Process Id     Burst Time   Wait Time    TurnAround Time\n");
    int wait_time=0;
    for(int i=0;i<n;i++)
    {
        printf("P%d         %d         %d         %d\n",index[i],b[i],wait_time,wait_time + b[i]);
        wait_time += b[i];
    }
    return 0;
}
```

Bankers Algorithm

```
#include <stdio.h>
#include <stdbool.h>

int main(){
    bool flag = true, end = false;
    int P, R, k = 0;
    printf("Enter no of processes, resources: ");
    scanf("%d%d", &P, &R);

    int ss[P], av[R], max[P][R], alloc[P][R], need[P][R];
    bool finished[100] = { false };
    printf("Enter no of resources available: ");
    for (int k = 0; k < R; k++) {
        scanf("%d", &av[k]);
    }

    for (int i = 0; i < P; i++) {
        printf("Enter max no of resources reqd by proc %d: ", i+1);
        for (int j = 0; j < R; j++) {
            scanf("%d", &max[i][j]);
```

```
        }

        printf("Enter no of resources allocated to proc %d: ", i+1);
        for (int j = 0; j < R; j++) {
            scanf("%d", &alloc[i][j]);
            need[i][j] = max[i][j] - alloc[i][j];
        }
    }

    while (!end) {
        for (int i = 0; i < P; i++) {
            if (av[0] >= *need[i] && finished[i] == false) {
                for (int j = 1; j < R; j++) {
                    if (need[i][j] > av[j]) {
                        flag = false;
                    }
                }
                if (flag) {
                    for (int j = 0; j < R; j++) {
                        av[j] += alloc[i][j];
                    }
                    ss[k] = i+1;
                    k++;
                    finished[i] = true;
                }
                end = true;
                for (int i = 0; i < P; i++) {
                    if (finished[i] == false) {
                        end = false;
                    }
                }
            }
        }
    }

    printf("\nSafety sequence: < ");
    for (int i = 0; i < P; i++) {
        printf("%d ", ss[i]);
    }
    printf(">\n");
}
```

Disk Scheduling (FCFS):

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

int seek_time(int arr[], int head, int len) {
    int seek_time = abs(arr[0] - head);
    for (int i = 1; i < len; i++)
```

```
        seek_time += abs(arr[i] - arr[i-1]);

    return seek_time;
}

int main() {
    int noOfReq, head;
    printf("Enter no of requests: ");
    scanf("%d", &noOfReq);

    int requests[noOfReq];
    printf("Enter sequence of requests:\n");
    for (int i = 0; i < noOfReq; i++)
        scanf("%d", &requests[i]);

    printf("Enter position of head: ");
    scanf("%d", &head);

    int ans = seek_time(requests, head, noOfReq);
    printf("Total seek time: %d\n", ans);
}
```

Disk Scheduling (Scan)

```
#include <stdio.h>
#include <stdlib.h>

int min(int arr[], int len) {
    int min = arr[0];
    for (int i = 1; i < len; i++) {
        if (arr[i] <= min) {
            min = arr[i];
        }
    }

    return min;
}

int main() {
    int noOfReq, head;
    printf("Enter no of requests: ");
    scanf("%d", &noOfReq);

    int requests[noOfReq], max_req = 0;
    printf("Enter sequence of requests:\n");
    for (int i = 0; i < noOfReq; i++) {
        scanf("%d", &requests[i]);
        max_req = (requests[i] >= max_req) ? requests[i] : max_req;
    }

    printf("Enter position of head: ");
    scanf("%d", &head);

    int range_min = 0, range_max = max_req + (10-1);
```

```
    int ans = abs(range_max - head) + (range_max - min(requests, noOfReq));
    printf("Total seek time: %d\n", ans);
}
```

Disk Scheduling (Look)

```
#include <stdio.h>
#include <stdlib.h>

int min(int arr[], int len) {
    int min = arr[0], index = 0;
    for (int i = 1; i < len; i++) {
        if (arr[i] <= min) {
            min = arr[i];
        }
    }

    return min;
}

int main() {
    int noOfReq, head;
    printf("Enter no of requests: ");
    scanf("%d", &noOfReq);

    int requests[noOfReq], max_req = 0;
    printf("Enter sequence of requests:\n");
    for (int i = 0; i < noOfReq; i++) {
        scanf("%d", &requests[i]);
        max_req = (requests[i] >= max_req) ? requests[i] : max_req;
    }

    printf("Enter position of head: ");
    scanf("%d", &head);

    int ans = abs(max_req - head) + (max_req - min(requests, noOfReq));
    printf("Total seek time: %d\n", ans);
}
```

Memory Allocation Strategy (Best Fit)

```
#include<stdio.h>

#define max 25
void main()
{
int frag[max],b[max],f[max],i,j,nb,nf,temp,lowest=10000;
static int bf[max],ff[max];

printf("\nEnter the number of blocks:");
scanf("%d",&nb);
printf("Enter the number of files:");
scanf("%d",&nf);
printf("\nEnter the size of the blocks:-\n");
for(i=1;i<=nb;i++)
{
printf("Block %d:",i);
scanf("%d",&b[i]);
}
printf("Enter the size of the files :-\n");
for(i=1;i<=nf;i++)
{
printf("File %d:",i);
scanf("%d",&f[i]);
}
for(i=1;i<=nf;i++)
{
for(j=1;j<=nb;j++)
{
if(bf[j]!=1)
{
temp=b[j]-f[i];
if(temp>=0)
if(lowest>temp)
{
ff[i]=j;

lowest=temp;
}
}
}
frag[i]=lowest;
bf[ff[i]]=1;
lowest=10000;
}
printf("\nFile No\tFile Size \tBlock No\tBlock Size\tFragment");
for(i=1;i<=nf && ff[i]!=0;i++)
printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);

}
```

Memory Allocation Strategy (Worst Fit)

```
#include<stdio.h>

#define max 25
void main()
{
int frag[max],b[max],f[max],i,j,nb,nf,temp;
static int bf[max],ff[max];

printf("\n\tMemory Management Scheme - First Fit");
printf("\nEnter the number of blocks:");
scanf("%d",&nb);
printf("Enter the number of files:");
scanf("%d",&nf);
printf("\nEnter the size of the blocks:-\n");
for(i=1;i<=nb;i++)
{
printf("Block %d:",i);
scanf("%d",&b[i]);
}
printf("Enter the size of the files :-\n");
for(i=1;i<=nf;i++)
{
printf("File %d:",i);
scanf("%d",&f[i]);
}
for(i=1;i<=nf;i++)
{
for(j=1;j<=nb;j++)
{
if(bf[j]!=1)
{
temp=b[j]-f[i];
if(temp>=0)
{
ff[i]=j;
break;
}
}
}
frag[i]=temp;
bf[ff[i]]=1;
}
printf("\nFile_no:\tFile_size :\tBlock_no:\tBlock_size:\tFragement");
for(i=1;i<=nf;i++)
printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);

}
```