CPU scheduling algorithms

a. FCFS

```
#include<stdio.h>
int main()
{
int bt[20], wt[20], tat[20], i, n;
float wtavg, tatavg;
printf("\nEnter the number of processes -- ");
scanf("%d", &n);
for(i=0;i<n;i++)
{
printf("\nEnter Burst Time for Process %d -- ", i);
scanf("%d", &bt[i]);
}
wt[0] = wtavg = 0;
tat[0] = tatavg = bt[0];
for(i=1;i<n;i++)
{
wt[i] = wt[i-1] +bt[i-1];
tat[i] = tat[i-1] +bt[i];
wtavg = wtavg + wt[i];
tatavg = tatavg + tat[i];
}
printf("\t PROCESS \tBURST TIME \t WAITING TIME\t TURNAROUND TIME\n");
for(i=0;i<n;i++)
printf("\n\t P%d \t\t %d \t\t %d \t\t %d", i, bt[i], wt[i], tat[i]);
printf("\nAverage Waiting Time -- %f\n", wtavg/n);
printf("\nAverage Turnaround Time -- %f\n", tatavg/n);
return 0;
}
```

b)SJF (Non-Pre-emptive)

```
#include<stdio.h>
int main()
{
int p[20], bt[20], wt[20], tat[20], i, k, n, temp; float wtavg, tatavg;
printf("\nEnter the number of processes -- ");
scanf("%d", &n);
for(i=0;i<n;i++)
{
p[i]=i;
printf("Enter Burst Time for Process %d -- ", i);
scanf("%d", &bt[i]);
}
for(i=0;i<n;i++)
for(k=i+1;k<n;k++)
if(bt[i]>bt[k])
{
temp=bt[i];
bt[i]=bt[k];
bt[k]=temp;
temp=p[i];
p[i]=p[k];
p[k]=temp;
}
wt[0] = wtavg = 0;
tat[0] = tatavg = bt[0]; for(i=1;i<n;i++)
{
wt[i] = wt[i-1] +bt[i-1];
tat[i] = tat[i-1] +bt[i];
wtavg = wtavg + wt[i];
tatavg = tatavg + tat[i];
}
printf("\n\t PROCESS \tBURST TIME \t WAITING TIME\t TURNAROUND TIME\n");
for(i=0;i<n;i++)
printf("\n\t P%d \t\t %d \t\t %d \t\t %d", p[i], bt[i], wt[i], tat[i]);
printf("\nAverage Waiting Time -- %f", wtavg/n);
printf("\nAverage Turnaround Time -- %f", tatavg/n);
return 0;
}
```

Tw 3:

Unix Process Control System calls

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

int main() {
   pid_t pid;

   // Fork a child process
   pid = fork();

   if (pid < 0) {
      // Fork failed
      fprintf(stderr, "Fork failed\n");
      return 1;
   }
   else if (pid == 0) {
      // Child process
      printf("Child process: My PID is %d\n", getpid());
      sleep(20);
      printf("Executing the child process ....\n");

      execlp("/bin/ls", "ls", NULL); // Executes 'ls' command
      // If execlp fails, this will run
      perror("execlp");
      exit(1);
   }
   else {
      // Parent process
      printf("Parent process: My PID is %d\n", getpid());
      printf("Child process sleeping for 20 seconds....\n");
      // Wait for the child to finish
      wait(NULL);
      printf("Parent process: Child process has terminated\n");
   }

   return 0;
}
```

Tw 4:

Process Synchronisation: The Dining Philosophers Problem.

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

int main() {
    int tph, i, howhung, cho;
    int philname[20], status[20], hu[20];

    printf("\n\nDINING PHILOSOPHER PROBLEM\n");
    printf("Enter the total no. of philosophers: ");
    scanf("%d", &tph);

    for (i = 0; i < tph; i++) {
        philname[i] = (i + 1);
        status[i] = 1; // 1 means thinking, 2 means hungry
    }

    printf("How many are hungry: ");
    scanf("%d", &howhung);

    if (howhung == tph) {
        printf("\nAll are hungry..\nDeadlock stage will occur");
        printf("\nExiting\n");
        exit(0);
    } else {
        for (i = 0; i < howhung; i++) {
            printf("Enter philosopher %d position (1 to %d): ", (i + 1), tph);
            scanf("%d", &hu[i]);
            if (hu[i] < 1 || hu[i] > tph) {
                printf("Invalid position! Exiting.\n");
                exit(0);
            }
            hu[i]--; // Adjust for zero-based indexing
            status[hu[i]] = 2; // Mark as hungry
        }

        do {
            printf("\n1. One can eat at a time\n2. Exit\nEnter your choice: ");
            scanf("%d", &cho);

            if (cho == 1) {
                int pos = 0, x;
                printf("\nAllow one philosopher to eat at any time\n");
                for (i = 0; i < howhung; i++, pos++) {
                    printf("\nP %d is granted to eat", philname[hu[pos]]);
                    for (x = pos + 1; x < howhung; x++) {
                        printf("\nP %d is waiting", philname[hu[x]]);
                    }
                }
            } else if (cho == 2) {
                exit(0);
            } else {
                printf("\nInvalid option..\n");
            }
        } while (1);
    }

    return 0;
}
```

Tw 5:

```
#include <stdio.h>
#include <semaphore.h>
#include <pthread.h>
#include <stdlib.h>
#include <time.h>

#define buffersize 10

pthread_mutex_t mutex;
pthread_t tidP[20], tidC[20];
sem_t full, empty;
int counter;
int buffer[buffersize];

void initialize() {
   pthread_mutex_init(&mutex, NULL);
   sem_init(&full, 0, 0);            // Correct pshared for threads
   sem_init(&empty, 0, buffersize);
   counter = 0;
}

void write_item(int item) {
   buffer[counter++] = item;
}

int read_item() {
   return buffer[--counter];
}

void *producer(void *param) {
   int item;
   item = rand() % 100;

   sem_wait(&empty);
   pthread_mutex_lock(&mutex);

   printf("\nProducer has produced item: %d\n", item);
   write_item(item);

   pthread_mutex_unlock(&mutex);
   sem_post(&full);

   return NULL;
}

void *consumer(void *param) {
   int item;

   sem_wait(&full);
   pthread_mutex_lock(&mutex);

   item = read_item();
```

```
    printf("\nConsumer has consumed item: %d\n", item);

    pthread_mutex_unlock(&mutex);
    sem_post(&empty);

    return NULL;
}

int main() {
    int n1, n2, i;

    srand(time(NULL));  // Initialize random seed
    initialize();

    printf("Enter the number of producers: ");
    scanf("%d", &n1);
    printf("Enter the number of consumers: ");
    scanf("%d", &n2);

    for (i = 0; i < n1; i++)
        pthread_create(&tidP[i], NULL, producer, NULL);

    for (i = 0; i < n2; i++)
        pthread_create(&tidC[i], NULL, consumer, NULL);

    for (i = 0; i < n1; i++)
        pthread_join(tidP[i], NULL);

    for (i = 0; i < n2; i++)
        pthread_join(tidC[i], NULL);

    return 0;
}
```

Tw 6:

Banker’s Program.

```
#include <stdio.h>

int main() {
    int numProcesses = 5; // Number of processes
    int numResources = 3; // Number of resources

    int allocationMatrix[5][3] = {{1, 1, 2}, {2, 1, 2}, {4, 0, 1}, {0, 2, 0}, {1, 1, 2}}; // Allocation Matrix
    int maxMatrix[5][3] = {{4, 3, 3}, {3, 2, 2}, {9, 0, 2}, {7, 5, 3}, {1, 1, 2}};   // MAX Matrix
    int availableResources[3] = {2, 1, 0}; // Available Resources

    int isFinished[numProcesses], safeSequence[numProcesses], index = 0;
    for (int k = 0; k < numProcesses; k++) {
        isFinished[k] = 0;
    }

    int needMatrix[numProcesses][numResources];
    for (int i = 0; i < numProcesses; i++) {
        for (int j = 0; j < numResources; j++)
            needMatrix[i][j] = maxMatrix[i][j] - allocationMatrix[i][j];
    }

    for (int k = 0; k < numProcesses; k++) {
        for (int i = 0; i < numProcesses; i++) {
            if (isFinished[i] == 0) {
                int flag = 0;
                for (int j = 0; j < numResources; j++) {
                    if (needMatrix[i][j] > availableResources[j]) {
                        flag = 1;
                        break;
                    }
                }
                if (flag == 0) {
                    safeSequence[index++] = i;
                    for (int y = 0; y < numResources; y++)
                        availableResources[y] += allocationMatrix[i][y];
                    isFinished[i] = 1;
                }
            }
        }
    }

    int flag = 1;
    for (int i = 0; i < numProcesses; i++) {
        if (isFinished[i] == 0) {
            flag = 0;
            printf("The system is not safe.\n");
            break;
        }
    }
    if (flag == 1) {
        printf("SAFE Sequence: ");
        for (int i = 0; i < numProcesses - 1; i++)
            printf("P%d -> ", safeSequence[i]);
        printf("P%d\n", safeSequence[numProcesses - 1]);
    }
    return 0;
}
```

Tw 9:

```
echo "the present working directory is"
pwd
echo "enter the directory name which you want to be a current working directory"

read dir

cd $dir

mkdir os_lab
echo " The directory named os_lab is created"

echo "enter the name of the directory to be removed"
read r
rmdir $r

echo "enter the source and destination file names"
read s1
read s2

cp $s1 $s2

mv $s1 $s2 os_lab
echo "both the files have moved to os_lab"
ls -l os_lab

echo "enter the name of the file to be removed"
read f
rm $f
echo "the file $f is removed successfully "

cd ..
echo "moving to root directory "
```

Tw 10:

```
echo "Enter the current working directory name"
read dir
ls -l > f2.txt
grep '^d' f2.txt > g1.txt
echo "The Number of Directories in the Current Working Directory is"
wc -l g1.txt
grep '^-' f2.txt > g1.txt
```

```
echo "The number of files in the current working directory is"
wc -l g1.txt
echo "Enter The Name of the file"
read file
ls -l $file

chmod u+x,g+w $file
echo " The file after changing the permission is"
ls -l $file
```