// FCFS

// #include<stdio.h>

// #include<stdlib.h>

// void main(){

// int n,p[30],bt[30],wt[30],tat[30],temp[30],at[30];

// float atat=0,awt=0;

// printf("Enter how many Process you want to enter:");

// scanf("%d",&n);

// printf("Enter the process number:");

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

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

// }

// printf("Enter the Arrival time of each Process:");

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

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

// }

// printf("Enter the Burst Time for each process:\n");

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

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

// }

// temp[0] = 0;

// printf("Process:\tArrival Time:\tBurst Time:\tWaiting Time:\tTurn Around Time:\n");

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

// wt[i] = 0;

// tat[i] = 0;

// temp[i+1] = temp[i]+bt[i];

// wt[i] = temp[i]-at[i];

// tat[i] = wt[i]+bt[i];

// awt = awt+wt[i];

// atat=atat+tat[i];

// printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\n",p[i],at[i],bt[i],wt[i],tat[i]);

// }

// awt = awt / n;

// atat = atat / n;

// printf("Average Waiting time is = %f\n",awt);

// printf("Average Turn Around time is = %f",atat);

// }

// SJF

// #include<stdio.h>

// void main(){

// int n,p[30],bt[30],wt[30],tat[30];

// float atat=0,awt=0;

// printf("Enter how many Process you want to enter:");

// scanf("%d",&n);

// printf("Enter the process number:");

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

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

// }

// printf("Enter the Burst Time for each process:\n");

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

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

// }

// // Applying Sorting

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

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

// if(bt[i]>bt[j]){

// int temp = bt[i];

// bt[i] = bt[j];

// bt[j] = temp;

// }

// }

// }

// printf("Process:\tBurst Time:\tWaiting Time:\tTurn Around Time:\n");

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

// wt[i] = 0;

// tat[i] = 0;

// for(int j=0;j<i;j++){

// wt[i] = wt[i]+bt[j];

// }

// tat[i] = wt[i]+bt[i];

// awt = awt+wt[i];

// atat=atat+tat[i];

// printf("%d\t\t%d\t\t%d\t\t%d\n",p[i],bt[i],wt[i],tat[i]);

// }

// awt = awt / n;

// atat = atat / n;

// printf("Average Waiting time is = %f\n",awt);

// printf("Average Turn Around time is = %f",atat);

// }

// FORK

// #include<stdio.h>

// #include<stdlib.h>

// #include<sys/types.h>

// #include<unistd.h>

// void main(){

// pid\_t value;

// value = fork();

// if(value<0){

// // Value is less than zero means unable to create a duplicate process.

// printf("Failed to create a duplicate process:");

// }

// else if(value==0){

// // Value is equals to zero means if child process is running

// printf("My Process Identification ID(Child) is = %d\n",getpid());

// printf("My Parent Process Identification ID is = %d\n",getppid());

// }

// else if(value>0){

// // Value is greater than zero means parent process is running.

// printf("My Child Process Identification ID is = %d\n",value);

// printf("My Process Identification ID(Parent) is = %d\n",getpid());

// }

// }

// ORPHAN

// #include<stdio.h>

// #include<stdlib.h>

// #include<sys/types.h>

// #include<unistd.h>

// void main(){

// pid\_t value;

// value = fork();

// if(value<0){

// // Value is less than zero means unable to create a duplicate process.

// printf("Failed to create a duplicate process:");

// }

// else if(value==0){

// // Value is equals to zero means if child process is running

// sleep(2);

// printf("I am Child and My Process Identification ID is = %d\n",getpid());

// printf("My Parent Process Identification ID is = %d\n",getppid());

// }

// else if(value>0){

// // Value is greater than zero means parent process is running.

// printf("I am Parent and My Process Identification ID is = %d\n",getpid());

// printf("My Child Process Identification ID is = %d\n",value);

// }

// }

// ZOMBIE

// #include<stdio.h>

// #include<stdlib.h>

// #include<sys/types.h>

// #include<unistd.h>

// void main(){

// pid\_t value;

// value = fork();

// if(value<0){

// // Value is less than zero means unable to create a duplicate process.

// printf("Failed to create a duplicate process:");

// }

// else if(value==0){

// // Value is equals to zero means if child process is running

// printf("I am Child and My Process Identification ID is = %d\n",getpid());

// printf("My Parent Process Identification ID is = %d\n",getppid());

// }

// else if(value>0){

// // Value is greater than zero means parent process is running.

// sleep(5);

// printf("I am Parent and My Process Identification ID is = %d\n",getpid());

// printf("My Child Process Identification ID is = %d\n",value);

// }

// }

// EXECL SYSTEM CALL

// #include<stdio.h>

// #include<unistd.h>

// void main(){

// printf("Before Execl System Call:\n");

// execl("/usr/bin/ls","ls",NULL);

// // Basically Execl System call will replace the current process image with ls command in the abaove code.

// printf("After:");

// }

// THREAD

// #include<stdio.h>

// #include<unistd.h>

// #include<pthread.h>

// // \*\*\* To complile this file include lpthread i.e. --> gcc fileName -lpthread

// void \*thread\_function(void \*args){

// printf("Inside Thread:\n");

// for(int j=20;j<25;j++){

// printf("%d\n",j);

// sleep(1);

// }

// }

// void main(){

// pthread\_t a\_thread;

// // pthread\_t is a datatype and a\_thread is a variable.

// pthread\_create(&a\_thread,NULL,thread\_function,NULL);

// // To create a thread we have to pass exactly 4 parameters to pthread\_create() function.

// // i.e. address\_Of\_the\_Thread\_Created,NULL means default syatem Attributes,Name of the thread function, NULL means no parameters passed to the function.

// pthread\_join(a\_thread,NULL);

// // This pthread\_join() function will wait to complete the execution of the thread function completely and then the control will back to the main function.

// // This function will take 2 parameters (i.e. for which thread it will wait and variable)

// printf("Inside Main Function:\n");

// for(int i=0;i<5;i++){

// printf("%d\n",i);

// sleep(1);

// }

// }