Task 2, task 3, task 4, task 5, task 6, task 7, task 8, task 9, task 10, task 11, task 12

TASK -2

FORK(), GETPID(), WAIT(), EXIT()

```
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>
int main() {
    int pid = fork();
    if (pid == 0) {
        printf("Child: PID = %d\n", getpid());
        _exit(0);
    } else {
        wait(NULL);
        printf("Parent: PID = %d\n", getpid());
    }
}
```

Output:

```
Child: PID = 1234
Parent: PID = 1233
```

EXEC() EXAMPLE

```
#include <stdio.h>
#include <unistd.h>
int main() {
    printf("Before exec\n");
    execl("/bin/ls", "ls", NULL);
    printf("This will not print if exec succeeds\n");
}
```

Output:
Lists current directory contents (like running ls).

CLOSE() EXAMPLE

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

```
#include <fcntl.h>
int main() {
    int fd = open("test.txt", O_CREAT|O_WRONLY, 0644);
    write(fd, "Hello", 5);
    close(fd);
    printf("File closed successfully\n");
}
```

STAT() EXAMPLE

```
#include <stdio.h>
#include <sys/stat.h>

int main() {
    struct stat s;
    stat("test.txt", &s);
    printf("File Size: %ld bytes\n", s.st_size);
}
```

OPENDIR() AND READDIR() EXAMPLE

```
#include <stdio.h>
#include <dirent.h>
int main() {
    DIR *d = opendir(".");
    struct dirent *de;
    while ((de = readdir(d)) != NULL)
        printf("%s\n", de->d_name);
    closedir(d);
}
```

Output:
Lists all files in the current directory.

TASK 3

.
SIMULATE CP COMMAND

Copies contents of one file to another.

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

```
int main() {
    FILE *src, *dest;
    char ch;
    src = fopen("source.txt", "r");
    dest = fopen("copy.txt", "w");
    while ((ch = fgetc(src)) != EOF)
        fputc(ch, dest);
    fclose(src);
    fclose(dest);
    printf("File copied successfully!\n");
    return 0;
}
```

Usage:
Create a file source.txt before running the program.
Output → creates copy.txt.

SIMULATE LS COMMAND

Lists all files in current directory.

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

int main() {
    struct dirent *d;
    DIR *dir = opendir(".");
    while ((d = readdir(dir)) != NULL)
        printf("%s\n", d->d_name);
    closedir(dir);
    return 0;
}
```

Output: Lists files (like ls).

SIMULATE GREP COMMAND

Searches for a word in a text file.

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

int main() {
   FILE *fp = fopen("data.txt", "r");
   char word[20] = "AI", line[100];
   while (fgets(line, sizeof(line), fp))
      if (strstr(line, word))
         printf("%s", line);
   fclose(fp);
   return 0;
}
```

Usage:
Create a file data.txt with few lines — any line containing "AI" will be printed.

TASK 4

FCFS:
```
#include <stdio.h>
int main() {
   int n = 4, i;
   int bt[] = {5, 3, 8, 6}, at[] = {0, 1, 2, 3};
   int ct[4], tat[4], wt[4];
   float awt = 0, atat = 0;
   ct[0] = at[0] + bt[0];
   tat[0] = ct[0] - at[0];
   wt[0] = tat[0] - bt[0];
   for (i = 1; i < n; i++) {
      ct[i] = (at[i] > ct[i-1]) ? at[i] + bt[i] : ct[i-1] + bt[i];
      tat[i] = ct[i] - at[i];
      wt[i] = tat[i] - bt[i];
   }
   printf("PID\tAT\tBT\tCT\tTAT\tWT\n");
   for (i = 0; i < n; i++) {
      printf("P%d\t%d\t%d\t%d\t%d\t%d\n", i+1, at[i], bt[i], ct[i], tat[i], wt[i]);
      atat += tat[i];
      awt += wt[i];
   }
```

```
    printf("\nAverage Turnaround Time = %.2f", atat/n);
    printf("\nAverage Waiting Time    = %.2f\n", awt/n);
}
```

SJF:

```
#include <stdio.h>
int main() {
    int n = 4, i, j, temp;
    int pid[] = {1, 2, 3, 4};
    int bt[] = {6, 8, 7, 3}, wt[4], tat[4];
    float awt = 0, atat = 0;
    for(i=0;i<n-1;i++)
        for(j=0;j<n-i-1;j++)
            if(bt[j] > bt[j+1]) {
                temp = bt[j]; bt[j] = bt[j+1]; bt[j+1] = temp;
                temp = pid[j]; pid[j] = pid[j+1]; pid[j+1] = temp;
            }
    wt[0] = 0;
    for(i=1;i<n;i++)
        wt[i] = wt[i-1] + bt[i-1];
    for(i=0;i<n;i++) {
        tat[i] = wt[i] + bt[i];
        awt += wt[i];
        atat += tat[i];
    }
    printf("PID\tBT\tWT\tTAT\n");
    for(i=0;i<n;i++)
        printf("P%d\t%d\t%d\t%d\n", pid[i], bt[i], wt[i], tat[i]);
    printf("\nAverage Waiting Time = %.2f", awt/n);
    printf("\nAverage Turnaround Time = %.2f\n", atat/n);
}
```

PRIORITY:

```
#include <stdio.h>
int main() {
    int n = 4, i, j, temp;
    int pid[] = {1, 2, 3, 4};
    int bt[] = {10, 1, 2, 1}, pr[] = {3, 1, 4, 2};
    int wt[4], tat[4];
    float awt = 0, atat = 0;
    for(i=0;i<n-1;i++)
        for(j=0;j<n-i-1;j++)
```

```
            if(pr[j] > pr[j+1]) {
                temp = pr[j]; pr[j] = pr[j+1]; pr[j+1] = temp;
                temp = bt[j]; bt[j] = bt[j+1]; bt[j+1] = temp;
                temp = pid[j]; pid[j] = pid[j+1]; pid[j+1] = temp;
            }
    wt[0] = 0;
    for(i=1;i<n;i++)
        wt[i] = wt[i-1] + bt[i-1];
    for(i=0;i<n;i++) {
        tat[i] = wt[i] + bt[i];
        awt += wt[i];
        atat += tat[i];
    }
    printf("PID\tBT\tPR\tWT\tTAT\n");
    for(i=0;i<n;i++)
        printf("P%d\t%d\t%d\t%d\t%d\n", pid[i], bt[i], pr[i], wt[i], tat[i]);
    printf("\nAverage Waiting Time = %.2f", awt/n);
    printf("\nAverage Turnaround Time = %.2f\n", atat/n);
}
```

RR:

```
#include <stdio.h>
int main() {
    int n = 4, tq = 3;
    int pid[] = {1, 2, 3, 4};
    int bt[] = {5, 4, 2, 1}, rt[4];
    int wt[4] = {0}, tat[4] = {0};
    float awt = 0, atat = 0;
    int time = 0, done;
    for (int i = 0; i < n; i++) rt[i] = bt[i];
    do {
        done = 1;
        for (int i = 0; i < n; i++) {
            if (rt[i] > 0) {
                done = 0;
                if (rt[i] > tq) {
                    time += tq;
                    rt[i] -= tq;
                } else {
                    time += rt[i];
                    wt[i] = time - bt[i];
                    rt[i] = 0;
                }
            }
        }
    } while (!done);
    for (int i = 0; i < n; i++) {
        tat[i] = bt[i] + wt[i];
        awt += wt[i];
        atat += tat[i];
    }
    printf("PID\tBT\tWT\tTAT\n");
    for (int i = 0; i < n; i++)
        printf("P%d\t%d\t%d\t%d\n", pid[i], bt[i], wt[i], tat[i]);
    printf("\nAverage Waiting Time = %.2f", awt / n);
    printf("\nAverage Turnaround Time = %.2f\n", atat / n);
}
```

TASK 5

SEMAPHORE:

```
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
#define MAX_PORTS 3   // maximum concurrent ports
#define THREADS 5     // total threads trying to access
sem_t port_sem;
void* access_port(void* arg) {
    int id = *(int*)arg;
    printf("Thread %d waiting for a port...\n", id);
    sem_wait(&port_sem); // acquire a port
    printf("Thread %d opened a port.\n", id);
    sleep(1); // simulate work
    printf("Thread %d closing port.\n", id);
    sem_post(&port_sem); // release port
    return NULL;
}
int main() {
    pthread_t t[THREADS];
    int id[THREADS];
    sem_init(&port_sem, 0, MAX_PORTS); // initialize semaphore
    for(int i=0;i<THREADS;i++){
        id[i]=i+1;
        pthread_create(&t[i], NULL, access_port, &id[i]);
    }
    for(int i=0;i<THREADS;i++)
        pthread_join(t[i], NULL);
    sem_destroy(&port_sem);
    return 0;
}
```

MONITORS:

```
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>
#define MAX_PORTS 3
#define THREADS 5
pthread_mutex_t mtx = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t cond = PTHREAD_COND_INITIALIZER;
int available_ports = MAX_PORTS;
void* access_port(void* arg) {
    int id = *(int*)arg;
    pthread_mutex_lock(&mtx);
    while(available_ports == 0)
        pthread_cond_wait(&cond, &mtx);
    available_ports--;
    printf("Thread %d opened a port. Available ports=%d\n", id, available_ports);
    pthread_mutex_unlock(&mtx);
    sleep(1); // simulate work
    pthread_mutex_lock(&mtx);
    available_ports++;
    printf("Thread %d closing port. Available ports=%d\n", id, available_ports);
    pthread_cond_signal(&cond);
    pthread_mutex_unlock(&mtx);
    return NULL;
}
int main() {
    pthread_t t[THREADS];
    int id[THREADS];
    for(int i=0;i<THREADS;i++){
        id[i]=i+1;
        pthread_create(&t[i], NULL, access_port, &id[i]);
    }
    for(int i=0;i<THREADS;i++)
        pthread_join(t[i], NULL);
    return 0;
}
```

TASK 6

PTHREAD. CONCURRENT:

```
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>

#define THREADS 3

void* run(void* arg) {
    int id = *(int*)arg;
    for(int i = 1; i <= 5; i++) {
        printf("Thread %d: iteration %d\n", id, i);
        usleep(100000); // sleep 0.1s to simulate work and allow context switching
    }
    return NULL;
}

int main() {
    pthread_t t[THREADS];
    int id[THREADS];

    for(int i = 0; i < THREADS; i++) {
        id[i] = i + 1;
        pthread_create(&t[i], NULL, run, &id[i]);
    }

    for(int i = 0; i < THREADS; i++)
        pthread_join(t[i], NULL);
    printf("Main thread: All threads finished.\n");
    return 0;
}
```

TASK 7

PRODUCER AND CONSUMER:

```
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
#define BUF_SIZE 5
#define ITEMS 10
int buffer[BUF_SIZE], in = 0, out = 0;
sem_t empty, full;
pthread_mutex_t m = PTHREAD_MUTEX_INITIALIZER;
void* producer(void* arg) {
    for(int i = 1; i <= ITEMS; i++) {
        sem_wait(&empty);           // wait for empty slot
        pthread_mutex_lock(&m);
        buffer[in] = i;
        in = (in + 1) % BUF_SIZE;
        printf("Produced: %d\n", i);
        pthread_mutex_unlock(&m);
        sem_post(&full);            // signal full slot
        usleep(100000);             // simulate production time
    }
    return NULL;
}
void* consumer(void* arg) {
    for(int i = 1; i <= ITEMS; i++) {
        sem_wait(&full);            // wait for full slot
        pthread_mutex_lock(&m);
        int val = buffer[out];
        out = (out + 1) % BUF_SIZE;
        printf("Consumed: %d\n", val);
        pthread_mutex_unlock(&m);
        sem_post(&empty);          // signal empty slot
        usleep(150000);            // simulate consumption time
    }
    return NULL;
}
int main() {
    pthread_t p, c;
    sem_init(&empty, 0, BUF_SIZE);
    sem_init(&full, 0, 0);
    pthread_create(&p, NULL, producer, NULL);
    pthread_create(&c, NULL, consumer, NULL);
    pthread_join(p, NULL);
    pthread_join(c, NULL);
    sem_destroy(&empty);
    sem_destroy(&full);
    printf("Producer-Consumer simulation done.\n");
    return 0;
}
```

TASK 8

FIRST FIT:

```
#include <stdio.h>
int main() {
    int mem[] = {100, 500, 200, 300, 600};
    int proc[] = {212, 417, 112, 426};
    int n = 5, m = 4;
    int alloc[4] = {-1,-1,-1,-1};
    for(int i=0;i<m;i++)
        for(int j=0;j<n;j++)
            if(proc[i]<=mem[j]) { alloc[i]=j; mem[j]-=proc[i]; break; }
    printf("Process\tSize\tBlock\n");
    for(int i=0;i<m;i++)
```

```
        printf("P%d\t\t%d\t\t%s\n", i+1, proc[i], alloc[i]==-1?"Not Allocated":(char[10]){48+alloc[i]});
}
```

WORST FIT:

```
#include <stdio.h>
int main() {
    int mem[] = {100, 500, 200, 300, 600};
    int proc[] = {212, 417, 112, 426};
    int n=5, m=4, alloc[4]={-1,-1,-1,-1};
    for(int i=0;i<m;i++){
        int w=-1;
        for(int j=0;j<n;j++)
            if(proc[i]<=mem[j] && (w==-1 || mem[j]>mem[w])) w=j;
        if(w!=-1){ alloc[i]=w; mem[w]-=proc[i]; }
    }
    printf("Process\tSize\tBlock\n");
    for(int i=0;i<m;i++)
        printf("P%d\t\t%d\t\t%s\n", i+1, proc[i], alloc[i]==-1?"Not Allocated":(char[10]){48+alloc[i]});
}
```

BEST FIT:

```
#include <stdio.h>
int main() {
    int mem[]={100,500,200,300,600};
    int proc[]={212,417,112,426};
    int n=5,m=4,alloc[4]={-1,-1,-1,-1};
    for(int i=0;i<m;i++){
        int b=-1;
        for(int j=0;j<n;j++)
            if(proc[i]<=mem[j] && (b==-1 || mem[j]<mem[b])) b=j;
        if(b!=-1){ alloc[i]=b; mem[b]-=proc[i]; }
    }
   printf("Process\tSize\tBlock\n");
    for(int i=0;i<m;i++)
        printf("P%d\t\t%d\t\t%s\n", i+1, proc[i], alloc[i]==-1?"Not Allocated":(char[10]){48+alloc[i]});
}
```

TASK 9

FIFO:

```
#include <stdio.h>

int main() {
    int pages[] = {1,2,3,4,1,2,5,1,2,3,4,5};
    int n = 12, f = 3, frame[3] = {-1,-1,-1}, i, j, k = 0, fault = 0;
    for(i=0;i<n;i++){
        int found=0;
        for(j=0;j<f;j++) if(frame[j]==pages[i]) found=1;
        if(!found){ frame[k]=pages[i]; k=(k+1)%f; fault++; }
        printf("%d -> ", pages[i]);
        for(j=0;j<f;j++) printf("%d ", frame[j]);
        printf("\n");
    }
    printf("Page Faults = %d\n", fault);
}
```

LRU:

```
#include <stdio.h>

int main(){
    int pages[]={1,2,3,4,1,2,5,1,2,3,4,5};
    int n=12,f=3,frame[3]={-1,-1,-1},used[3]={0},i,j,k,least,fault=0,t=0;
    for(i=0;i<n;i++){
        int found=0;
        for(j=0;j<f;j++)
            if(frame[j]==pages[i]){ found=1; used[j]=++t; }
        if(!found){
            int min=0;
            for(j=1;j<f;j++) if(used[j]<used[min]) min=j;
            frame[min]=pages[i]; used[min]=++t; fault++;
        }
        printf("%d -> ",pages[i]);
        for(j=0;j<f;j++) printf("%d ",frame[j]);
        printf("\n");
    }
    printf("Page Faults = %d\n",fault);
}
```

LFU:

```
#include <stdio.h>

int main(){
    int pages[]={1,2,3,4,1,2,5,1,2,3,4,5};
    int n=12,f=3,frame[3]={-1,-1,-1},freq[3]={0},i,j,k,fault=0;
    for(i=0;i<n;i++){
        int found=0;
        for(j=0;j<f;j++)
            if(frame[j]==pages[i]){ found=1; freq[j]++; }
        if(!found){
            int min=0;
            for(j=1;j<f;j++) if(freq[j]<freq[min]) min=j;
            frame[min]=pages[i]; freq[min]=1; fault++;
        }
        printf("%d -> ",pages[i]);
        for(j=0;j<f;j++) printf("%d ",frame[j]);
        printf("\n");
    }
    printf("Page Faults = %d\n",fault);
}
```

TASK 10

PAGING.C

```
#include <stdio.h>
int main() {
    int pm, lm, ps, frames, pages, i;
    printf("Enter the Size of Physical memory: ");
    scanf("%d", &pm);
    printf("Enter the size of Logical memory: ");
    scanf("%d", &lm);
    printf("Enter the partition size: ");
    scanf("%d", &ps);

    frames = pm / ps;  pages = lm / ps;
    printf("The physical memory is divided into %d no.of frames\n", frames);
    printf("The Logical memory is divided into %d no.of pages\n", pages);

    int pageTable[20], frameTable[50];
    for(i=0;i<frames;i++) frameTable[i]=32555;

    for(i=0;i<pages;i++){
        printf("Enter the Frame number where page %d must be placed: ", i);
        scanf("%d", &pageTable[i]);
        frameTable[pageTable[i]] = i;
    }

    printf("\nPAGE TABLE\nPage\tFrame\tPresence\n");
    for(i=0;i<pages;i++) printf("%d\t%d\t1\n", i, pageTable[i]);

    printf("\nFRAME TABLE\nFrame\tPage\n");
    for(i=0;i<frames;i++) printf("%d\t%d\n", i, frameTable[i]);

    int base, logAddr;
    printf("\nEnter Base Address: ");
    scanf("%d", &base);
    printf("Enter Logical Address: ");
    scanf("%d", &logAddr);

    int page = logAddr / ps, off = logAddr % ps;
    int phy = base + pageTable[page]*ps + off;
    printf("The Physical Address where the instruction present: %d\n", phy);
}
```

SEGMENTATION:

```
#include <stdio.h>
struct seg { int base, limit, val[10]; } s[10];
int main() {
    int n, i, j, seg, off;
    printf("Enter the size of the segment table: ");
    scanf("%d", &n);
    for(i=0;i<n;i++){
        printf("Enter info for segment %d\n", i+1);
        printf("Base: "); scanf("%d",&s[i].base);
        printf("Limit: "); scanf("%d",&s[i].limit);
        for(j=0;j<s[i].limit;j++){
            printf("Enter value at %d: ", s[i].base+j);
            scanf("%d",&s[i].val[j]);
        }
    }
    printf("\nSEG.NO\tBASE\tLIMIT\n");
    for(i=0;i<n;i++)
        printf("%d\t%d\t%d\n", i+1, s[i].base, s[i].limit);

    char ch='y';
    while(ch=='y'||ch=='Y'){
        printf("\nEnter logical address (segment offset): ");
        scanf("%d%d",&seg,&off);
        seg--;
        if(seg>=n||off>=s[seg].limit) printf("Invalid!\n");
        else
            printf("Logical=%d%d, Physical=%d, Value=%d\n",
                   seg+1,off,s[seg].base+off,s[seg].val[off]);
        printf("Continue(Y/N)? "); scanf(" %c",&ch);
    }
}
```

TASK 11

BANKERS:

```
#include <stdio.h>
int main() {
    int n=5, m=3, i, j, k, y=0;
    int alloc[5][3]={{0,1,0},{2,0,0},{3,0,2},{2,1,1},{0,0,2}};
    int max[5][3]  ={{7,5,3},{3,2,2},{9,0,2},{2,2,2},{4,3,3}};
    int avail[3]={3,3,2}, f[5]={0}, ans[5], need[5][3];

    for(i=0;i<n;i++)
        for(j=0;j<m;j++)
            need[i][j]=max[i][j]-alloc[i][j];

    for(k=0;k<n;k++)
        for(i=0;i<n;i++)
            if(f[i]==0){
                int flag=0;
            for(j=0;j<m;j++)
                if(need[i][j]>avail[j]){ flag=1; break; }
            if(!flag){
                ans[y++]=i;
                for(j=0;j<m;j++) avail[j]+=alloc[i][j];
                f[i]=1;
            }
        }

    printf("Safe Sequence: ");
    for(i=0;i<n-1;i++) printf("P%d -> ",ans[i]);
    printf("P%d\n",ans[n-1]);
}
```

TASK 12

Task-12

a)Sequential

```
#include <stdio.h>
int main() {
    int n=3, start[]={0,10,20}, len[]={5,3,4};
    for(int i=0;i<n;i++)
        printf("File %d: Blocks %d to %d\n", i+1, start[i], start[i]+len[i]-1);
}
```

DYNAMIC:

```
#include <stdio.h>
int main() {
    int n=3, index[]={1,4,7}, blocks[3][3]={{2,3,4},{5,6,7},{8,9,10}};
    for(int i=0;i<n;i++){
        printf("File %d -> Index Block %d -> ", i+1, index[i]);
        for(int j=0;j<3;j++) printf("%d ", blocks[i][j]);
        printf("\n");
    }
}
```

LINKED:

```
#include <stdio.h>
int main() {
    int n=3;
    int files[3][3]={{1,2,3},{4,5,6},{7,8,9}};
    for(int i=0;i<n;i++){
        printf("File %d: ", i+1);
        for(int j=0;j<3;j++)
            printf("%d -> ", files[i][j]);
        printf("NULL\n");
    }
}
```