1.Process ID,wait,waitpid

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

int wait_func() {
   int pid_1 = fork();
   if (pid_1 == 0) {
      printf("Process pid_1 successfully created\n");
      printf("Process pid_1 id: %d\n", getpid());
      exit(0);
   }
   waitpid(pid_1, NULL, 0);
   printf("pid_1 process terminated.\n");
   return 0;
}

int main() {
   int pid = fork();
   if (pid == 0) {
      printf("Child process created\n");
      printf("Current child is: %d\n", getppid());
      printf("Current parent is: %d\n", getpid());
      exit(0);
   }
   wait(NULL);
   printf("Child process is terminated.\n");
   wait_func();
   return 0;
}
```

2. Process Scheduling FCFS (Non-preemptive)

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

typedef struct {
   int at;
   int bt;
   int pid;
} Process;

int compare(const void* a, const void* b) {
   Process* pa = (Process*)a;
   Process* pb = (Process*)b;
   return (pa->at - pb->at);
}

int main() {
   int n;
   printf("Enter Number of processes: ");
   scanf("%d", &n);
   Process at_bt[n];
   int i;
   printf("Enter arrival time, burst time and process id:\n");
   for (i = 0; i < n; i++) {
      scanf("%d %d %d", &at_bt[i].at, &at_bt[i].bt, &at_bt[i].pid);
   }
   qsort(at_bt, n, sizeof(Process), compare);
   int ct[n];
   int tat[n];
   int wt[n];
   double total_tat = 0;
```

```
    double total_wt = 0;
    int count = at_bt[0].at;
    for (i = 0; i < n; i++) {
        if (at_bt[i].at <= count) {
            count += at_bt[i].bt;
            ct[i] = count;
            tat[i] = ct[i] - at_bt[i].at;
            total_tat += tat[i];
            wt[i] = tat[i] - at_bt[i].bt;
            total_wt += wt[i];
        } else {
            count++;
            i--;
        }
    }
    printf("\n%-5s%-5s%-5s%-5s%-5s%-5s\n", "PID", "AT", "BT", "CT", "TAT", "WT");
    for (i = 0; i < n; i++) {
        printf("%-5d%-5d%-5d%-5d%-5d%-5d\n", at_bt[i].pid, at_bt[i].at,
                at_bt[i].bt, ct[i], tat[i], wt[i]);
    }
    printf("Avg WT : %.2f\n", (total_wt / n));
    printf("Avg TAT : %.2f\n", (total_tat / n));
    return 0;
}
```

3. Process Scheduling SJF(Preemptive)

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

struct Process {
    int pid;
    int bt;
```

```
    int art;
};

void findWaitingTime(struct Process proc[], int n, int wt[]) {
    int rt[n];
    for (int i = 0; i < n; i++)
        rt[i] = proc[i].bt;
    int complete = 0, t = 0, minm = INT_MAX;
    int shortest = 0, finish_time;
    int check = 0;
    while (complete != n) {
        for (int j = 0; j < n; j++) {
            if ((proc[j].art <= t) && (rt[j] < minm) && rt[j] > 0) {
                minm = rt[j];
                shortest = j;
                check = 1;
            }
        }
        if (check == 0) {
            t++;
            continue;
        }
        rt[shortest]--;
        minm = rt[shortest];
        if (minm == 0)
            minm = INT_MAX;
        if (rt[shortest] == 0) {
            complete++;
            check = 0;
            finish_time = t + 1;
            wt[shortest] = finish_time - proc[shortest].bt - proc[shortest].art;
```

```
        if (wt[shortest] < 0)
            wt[shortest] = 0;
      }
      t++;
   }
}

void findTurnAroundTime(struct Process proc[], int n, int wt[], int tat[]) {
   for (int i = 0; i < n; i++)
      tat[i] = proc[i].bt + wt[i];
}

void findavgTime(struct Process proc[], int n) {
   int wt[n], tat[n], total_wt = 0, total_tat = 0;
   findWaitingTime(proc, n, wt);
   findTurnAroundTime(proc, n, wt, tat);
   printf(" P\t\tBT\t\tWT\t\tTAT\t\t\n");
   for (int i = 0; i < n; i++) {
      total_wt += wt[i];
      total_tat += tat[i];
      printf(" %d\t\t%d\t\t %d\t\t %d\n", proc[i].pid, proc[i].bt, wt[i], tat[i]);
   }
   printf("\nAverage waiting time = %.2f", (float)total_wt / (float)n);
   printf("\nAverage turn around time = %.2f", (float)total_tat / (float)n);
}

int main() {
   int n;
   printf("Enter no. of processes : ");
   scanf("%d", &n);
   struct Process proc[n];
```

```
    printf("Enter process id, arrival time and burst times : \n");
    int a, b, c;
    for (int i = 0; i < n; i++) {
        scanf("%d %d %d", &a, &b, &c);
        proc[i].pid = a;
        proc[i].art = b;
        proc[i].bt = c;
    }
    findavgTime(proc, n);
    return 0;
}
```

4.Producer Consumer Problem

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

int full = 0;
int empty = 10;
int x = 0;
int mutex = 1;

void producer() {
    mutex--;
    full++;
    empty--;
    x++;
    printf("PRODUCER PRODUCES THE ITEM %d \n", x);
    mutex++;
}

void consumer() {
    mutex--;
```

```
    full--;
    empty++;
    printf("CONSUMER CONSUMES THE ITEM %d \n" , x);
    x--;
    mutex++;
}

int main() {
    int n;

    while(1) {
        printf("1. PRESS \"1\" FOR PRODUCER\n");
        printf("2. PRESS \"2\" FOR CONSUMER\n");
        printf("3. PRESS \"3\" FOR EXIT\n");
        printf("ENTER THE CHOICE: ");
        scanf("%d" , &n);

        switch(n) {
            case 1:
                if(mutex == 1 && empty != 0) {
                    producer();
                }
                else {
                    printf("BUFFER IS FULL\n");
                }
                break;

            case 2:
                if(mutex == 1 && full != 0) {
                    consumer();
                }
```

```
            else {
               printf("BUFFER IS EMPTY\n");
            }
            break;

         case 3:
            printf("\nAditya Dikonda\nS12\n24\n");
            exit(0);
      }
   }

   return 0;
}
```

5. Banker's Algorithm

```
#include<stdio.h>

int main() {
   int p, c, count = 0, i, j, alc[5][3], max[5][3], need[5][3], safe[5], available[3], finish[5], terminate =
0;

   printf("Enter the number of processes and resources: ");
   scanf("%d %d", &p, &c);

   printf("Enter allocation of resources for all processes (%d x %d matrix):\n", p, c);
   for (i = 0; i < p; i++) {
      for (j = 0; j < c; j++) {
         scanf("%d", &alc[i][j]);
      }
   }

   printf("Enter the maximum resources required by each process (%d x %d matrix):\n", p, c);
```

```
for (i = 0; i < p; i++) {
   for (j = 0; j < c; j++) {
      scanf("%d", &max[i][j]);
   }
}

printf("Enter the available resources: ");
for (i = 0; i < c; i++) {
   scanf("%d", &available[i]);
}

printf("\nNeed resources matrix:\n");
for (i = 0; i < p; i++) {
   for (j = 0; j < c; j++) {
      need[i][j] = max[i][j] - alc[i][j];
      printf("%d\t", need[i][j]);
   }
   printf("\n");
}

for (i = 0; i < p; i++) {
   finish[i] = 0;
}

while (count < p) {
   for (i = 0; i < p; i++) {
      if (finish[i] == 0) {
         for (j = 0; j < c; j++) {
            if (need[i][j] > available[j]) {
               break;
            }
```

```
            }
            if (j == c) {
               safe[count] = i;
               finish[i] = 1;
               for (j = 0; j < c; j++) {
                  available[j] += alc[i][j];
               }
               count++;
               terminate = 0;
            } else {
               terminate++;
            }
         }
      }
      if (terminate == (p - 1)) {
         printf("Safe sequence does not exist\n");
         break;
      }
   }

   if (terminate != (p - 1)) {
      printf("\nAvailable resources after completion:\n");
      for (i = 0; i < c; i++) {
         printf("%d\t", available[i]);
      }
      printf("\nSafe sequence:\n");
      for (i = 0; i < p; i++) {
         printf("p%d\t", safe[i]);
      }
      printf("\n");
   }
```

```
    return 0;
}
```

6. Dining Philosopher's Problem

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

sem_t room;
sem_t chopstick[5];

void *philosopher(void *);
void eat(int);

int main() {
    int i, a[5];
    pthread_t tid[5];

    sem_init(&room, 0, 4);
    for (i = 0; i < 5; i++)
        sem_init(&chopstick[i], 0, 1);

    for (i = 0; i < 5; i++) {
        a[i] = i;
        pthread_create(&tid[i], NULL, philosopher, (void *)&a[i]);
    }

    for (i = 0; i < 5; i++)
        pthread_join(tid[i], NULL);
```

```
    return 0;
}

void *philosopher(void *num) {
    int phil = *(int *)num;
    sem_wait(&room);
    printf("\nPhilosopher %d has entered the room", phil);
    sem_wait(&chopstick[phil]);
    sem_wait(&chopstick[(phil + 1) % 5]);
    eat(phil);
    sleep(2);
    printf("\nPhilosopher %d has finished eating", phil);
    sem_post(&chopstick[(phil + 1) % 5]);
    sem_post(&chopstick[phil]);
    sem_post(&room);
}

void eat(int phil) {
    printf("\nPhilosopher %d is eating", phil);
}
```

7. Memory allocation techniques(Best fit, Worst fit, First fit)

```
#include<stdio.h>

void bestFit(int blockSize[], int m, int processSize[], int n) {
    int allocation[n], i, j;
    for (i = 0; i < n; i++)
        allocation[i] = -1;
    for (i = 0; i < n; i++) {
        int bestIdx = -1;
        for (j = 0; j < m; j++) {
```

```
            if (blockSize[j] >= processSize[i]) {
                if (bestIdx == -1)
                    bestIdx = j;
                else if (blockSize[bestIdx] > blockSize[j])
                    bestIdx = j;
            }
        }
        if (bestIdx != -1) {
            allocation[i] = bestIdx;
            blockSize[bestIdx] -= processSize[i];
        }
    }
    printf("\nProcess No.\tProcess Size\tBlock no.\n");
    for (i = 0; i < n; i++) {
        printf("%d\t\t%d\t\t", i+1, processSize[i]);
        if (allocation[i] != -1)
            printf("%d\n", allocation[i] + 1);
        else
            printf("Not Allocated\n");
    }
}

void firstFit(int blockSize[], int m, int processSize[], int n) {
    int allocation[n], i, j;
    for (i = 0; i < n; i++)
        allocation[i] = -1;
    for (i = 0; i < n; i++) {
        for (j = 0; j < m; j++) {
            if (blockSize[j] >= processSize[i]) {
                allocation[i] = j;
                blockSize[j] -= processSize[i];
```

```
            break;
        }
      }
   }
   printf("\nProcess No.\tProcess Size\tBlock no.\n");
   for (i = 0; i < n; i++) {
      printf("%d\t\t%d\t\t", i+1, processSize[i]);
      if (allocation[i] != -1)
         printf("%d\n", allocation[i] + 1);
      else
         printf("Not Allocated\n");
   }
}

void worstFit(int blockSize[], int m, int processSize[], int n) {
   int allocation[n], i, j;
   for (i = 0; i < n; i++)
      allocation[i] = -1;
   for (i = 0; i < n; i++) {
      int wstIdx = -1;
      for (j = 0; j < m; j++) {
         if (blockSize[j] >= processSize[i]) {
            if (wstIdx == -1)
               wstIdx = j;
            else if (blockSize[wstIdx] < blockSize[j])
               wstIdx = j;
         }
      }
      if (wstIdx != -1) {
         allocation[i] = wstIdx;
         blockSize[wstIdx] -= processSize[i];
```

```
        }
    }
    printf("\nProcess No.\tProcess Size\tBlock no.\n");
    for (i = 0; i < n; i++) {
        printf("%d\t\t%d\t\t", i+1, processSize[i]);
        if (allocation[i] != -1)
            printf("%d\n", allocation[i] + 1);
        else
            printf("Not Allocated\n");
    }
}

int main() {
    int blockSize[] = {100, 500, 200, 300, 600};
    int processSize[] = {212, 417, 112, 426};
    int m = sizeof(blockSize)/sizeof(blockSize[0]);
    int n = sizeof(processSize)/sizeof(processSize[0]);

    printf("First Fit:\n");
    firstFit(blockSize, m, processSize, n);

    printf("\nBest Fit:\n");
    bestFit(blockSize, m, processSize, n);

    printf("\nWorst Fit:\n");
    worstFit(blockSize, m, processSize, n);

    return 0;
}
```

8. FIFO,LRU

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

```
#define MAX 30

void display(int arr[], int n)
{
   int i;
   for (i = 0; i < n; i++)
   {
      printf("%d ", arr[i]);
   }
   printf("\n");
}

void fifo(int pages[], int n, int capacity)
{
   int frame[MAX], i, j, k, flag = 0, count = 0;
   for (i = 0; i < capacity; i++)
      frame[i] = -1;
   j = 0;
   printf("\nReference string|\tPage Frames\n");
   for (i = 0; i < n; i++)
   {
      printf("%d\t\t|\t", pages[i]);
      for (k = 0; k < capacity; k++)
         if (frame[k] == pages[i])
            flag = 1;
      if (flag == 0)
      {
         frame[j] = pages[i];
         j = (j + 1) % capacity;
         count++;
```

```
            display(frame, capacity);
        }
        else
        {
            flag = 0;
            display(frame, capacity);
        }
    }
    printf("\nNumber of page faults = %d\n", count);
}

int findLRU(int time[], int n)
{
    int i, minimum = time[0], pos = 0;
    for (i = 1; i < n; ++i)
    {
        if (time[i] < minimum)
        {
            minimum = time[i];
            pos = i;
        }
    }
    return pos;
}

void lru(int pages[], int n, int capacity)
{
    int frame[MAX], age[MAX] = {0}, i, k, flag = 0, count = 0, least;
    for (i = 0; i < capacity; i++)
        frame[i] = -1;
    printf("\nReference string|\tPage Frames\n");
```

```
    for (i = 0; i < n; i++)
    {
        printf("%d\t\t|\t", pages[i]);
        for (k = 0; k < capacity; k++)
            if (frame[k] == pages[i])
            {
                flag = 1;
                age[k] = i + 1;
            }
        if (flag == 0)
        {
            least = findLRU(age, capacity);
            age[least] = i + 1;
            frame[least] = pages[i];
            count++;
            display(frame, capacity);
        }
        else
        {
            flag = 0;
            display(frame, capacity);
        }
    }
    printf("\nNumber of page faults = %d\n", count);
}

int main()
{
    int pages[MAX], np, nf, i;
    printf("Enter the number of pages: ");
    scanf("%d", &np);
```

```
    printf("Enter the reference string values:\n");
    for (i = 0; i < np; i++)
        scanf("%d", &pages[i]);
    printf("Enter the number of frames: ");
    scanf("%d", &nf);
    fifo(pages, np, nf);
    lru(pages, np, nf);
    return 0;
}
```

9. FCFS,SCAN,C-SCAN

```
#include<stdio.h>
#include<stdlib.h>
#define MAX 20

void sort(int queue[], int n) {
    int i, j, temp;
    for(i = 0; i < n; i++) {
        for(j = i + 1; j < n; j++) {
            if(queue[i] > queue[j]) {
                temp = queue[i];
                queue[i] = queue[j];
                queue[j] = temp;
            }
        }
    }
}

void fcfs(int head, int queue[], int n) {
    int i, seek_count = 0;
    for(i = 0; i < n; i++) {
        seek_count += abs(head - queue[i]);
```

```
        head = queue[i];
    }
    printf("Total number of seek operations = %d\n", seek_count);
}

void scan(int head, int queue[], int n) {
    int i, seek_count = 0, distance, cur_track;
    sort(queue, n);
    for (i = 0; i < n; i++) {
        cur_track = queue[i];
        distance = abs(cur_track - head);
        seek_count += distance;
        head = cur_track;
    }
    printf("Total number of seek operations = %d\n", seek_count);
}

void cscan(int head, int queue[], int n) {
    int i, seek_count = 0, distance, cur_track, max = 200;
    sort(queue, n);
    seek_count = abs(head - 0);
    seek_count += abs(max - queue[0]);
    printf("Total number of seek operations = %d\n", seek_count);
}

int main() {
    int queue[MAX], n, i, head;
    printf("Enter the number of disk locations: ");
    scanf("%d", &n);
    printf("Enter the disk locations to read: ");
    for(i = 0; i < n; i++) {
```

```
        scanf("%d", &queue[i]);
    }
    printf("Enter the initial head position: ");
    scanf("%d", &head);
    printf("\nFCFS Disk Scheduling:\n");
    fcfs(head, queue, n);
    printf("\nSCAN Disk Scheduling:\n");
    scan(head, queue, n);
    printf("\nC-SCAN Disk Scheduling:\n");
    cscan(head, queue, n);
    return 0;
}
```