ARAVALI COLLEGE OF ENGINEERING & MANAGEMENT

Jasana Tigoan Road Greater Faridabad Haryana,
121006

OPERATING SYSTEM

LAB FILE

DEPARTMENT OF COMPUTER SCIENCE &

ENGINEERING (2024-2025)

FACULTY INCHARGE **Submitted By:**

Ms Priyanka **Name: ANKIT KUMAR**

(Assistant Professor) **Roll No: 24011312010**

S.no	Experiment Title	Date	Page	Signature
1	Write C program to demonstrate various process related concept.			
2	Write C programs to demonstrate various thread related concepts:-			
3	Write C Programs to simulate CPU scheduling algorithms: FCFS,SJF and RoundRobin.			
4	Write C programs to simulate Intra & Inter process communication (IPC) techniques: Pipes, Message Queues & Shared Memory.			
5	Write C program to stimulate solutions to classical process synchronization problems: Dining Philosopher, Producer – Consumer, Reader-Writer.			
6	Write C program to simulate Banker‟s Algorithm for Deadlock avoidance			
7	Write C program to simulate Page Replacement Algorithms: FIFO, LRU			
8	Write C program to simulate implementation of Disk Scheduling Algorithms: FCFS, SSTF			
9	Write C programs to implement UNIX System Calls and File Management.			

EXPERIMENT NO. 1

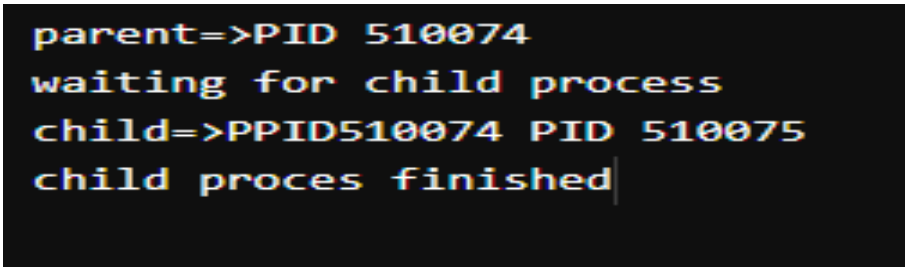
AIM: Write a program for process related concept (fork).

Code:

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

int main()
{
    int pid=fork();
    if(pid==0)
   {
       printf("child=>PPID%d PID %d\n",getppid(),getpid());
       exit(EXIT_SUCCESS);
   }
  else if(pid>0)
{
    printf("parent=>PID %d\n",getpid());
    printf("waiting for child process\n");
    wait(NULL);
    printf("child proces finished");
}
else{
    printf("unable to create child process");
}
    return EXIT_SUCCESS;
}
```

Output:



EXPERIMENT NO. 2

AIM: Write a program for various thread concepts.

Code:

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

void* routine() {
    printf("test from threads \n");
    printf("test from threads \n");
    printf("Ending \n");
    return NULL;
}

int main() {
    pthread_t t1, t2;
    pthread_create(&t1, NULL, &routine, NULL);
    pthread_create(&t2, NULL, &routine, NULL);

    pthread_join(t1, NULL);
    pthread_join(t2, NULL);

    return 0;
}
```

Output:

```
test from threads
test from threads
Ending
test from threads
test from threads
Ending
```

EXPERIMENT NO. 3

AIM: Write C programs to simulate CPU scheduling algorithms: FCFS, SJF, and Round Robin.

(a) FCFS Scheduling

```
#include <stdio.h>

int main() {
   int n, bt[^20], wt[^20] = {0}, tat[^20], i;
   float avwt = 0, avtat = 0;

   printf("Enter number of processes: ");
   scanf("%d", &n);

   printf("Enter burst times:\n");
   for(i = 0; i < n; i++) scanf("%d", &bt[i]);

   for(i = 1; i < n; i++) wt[i] = wt[i - 1] + bt[i - 1];

   printf("\nProcess\tBT\tWT\tTAT\n");
   for(i = 0; i < n; i++) {
      tat[i] = bt[i] + wt[i];
      avwt += wt[i];
      avtat += tat[i];
      printf("P%d\t%d\t%d\t%d\n", i + 1, bt[i], wt[i], tat[i]);
   }

   printf("\nAvg WT = %.2f\nAvg TAT = %.2f\n", avwt / n, avtat / n);
   return 0;
}
```

Output Example:

```
Enter number of processes: 3
Enter burst times:
5 3 8

Process BT   WT   TAT
P1       5   0    5
P2       3   5    8
P3       8   8    16

Avg WT = 4.33
Avg TAT = 9.67
```

(b) SJF Scheduling

```
#include <stdio.h>

int main() {
   int n, i, j, temp;
   printf("Enter number of processes: ");
   scanf("%d", &n);

   int bt[n], wt[n] = {0}, tat[n], p[n];
   float total_wt = 0, total_tat = 0;

   printf("Enter burst times:\n");
   for(i = 0; i < n; i++) {
      printf("P%d: ", i + 1);
      scanf("%d", &bt[i]);
      p[i] = i + 1;
   }

   for(i = 0; i < n-1; i++)
      for(j = i+1; j < n; j++)
         if(bt[i] > bt[j]) {
```

```
            temp = bt[i]; bt[i] = bt[j]; bt[j] = temp;
            temp = p[i]; p[i] = p[j]; p[j] = temp;
        }

    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];
        total_wt += wt[i];
        total_tat += tat[i];
    }

    printf("\nProcess\tBT\tWT\tTAT\n");
    for(i = 0; i < n; i++)
        printf("P%d\t%d\t%d\t%d\n", p[i], bt[i], wt[i], tat[i]);

    printf("\nAverage WT = %.2f\nAverage TAT = %.2f\n", total_wt / n, total_tat / n);

    return 0;
}
```

Output Example:

```
Enter number of processes: 3
Enter burst times:
P1: 5
P2: 3
P3: 8

Process BT WT TAT
P2      3  0  3
P1      5  3  8
P3      8  8  16

Average WT = 3.67    Average TAT = 9.00
```

(c) Round Robin Scheduling

```
#include <stdio.h>

int main() {
    int n, tq, bt[^20], rt[^20], wt[^20], tat[^20], time = 0;
    float total_wt = 0, total_tat = 0;

    printf("Enter total number of processes: ");
    scanf("%d", &n);

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

    printf("Enter Time Quantum: ");
    scanf("%d", &tq);

    while (1) {
        int done = 1;
        for (int i = 0; i < n; i++) {
            if (rt[i] > 0) {
                done = 0;
                int time_slice = (rt[i] > tq) ? tq : rt[i];
                time += time_slice;
                rt[i] -= time_slice;
                if (rt[i] == 0) wt[i] = time - bt[i];
            }
        }
        if (done) break;
    }

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

```
        tat[i] = bt[i] + wt[i];
        total_wt += wt[i];
        total_tat += tat[i];
    }

    printf("\nProc\tBT\tWT\tTAT");
    for (int i = 0; i < n; i++) {
        printf("\nP[%d]\t%d\t%d\t%d", i + 1, bt[i], wt[i], tat[i]);
    }
    printf("\n\nAverage Waiting Time: %.2f", total_wt / n);
    printf("\nAverage Turnaround Time: %.2f\n", total_tat / n);

    return 0;
}
```

Output Example:

```
Enter total number of processes: 3
Enter burst time for each process:
P[^1]: 5
P[^2]: 3
P[^3]: 8
Enter Time Quantum: 2

Proc    BT  WT  TAT
P[^1]    5   6   11
P[^2]    3   4   7
P[^3]    8   7   15

Average Waiting Time: 5.67
Average Turnaround Time: 11.00
```

EXPERIMENT NO. 4

AIM: Write C programs to simulate Intra & Inter process communication (IPC) techniques: Pipes, Message Queues & Shared Memory.

(a) Shared Memory

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

int main() {
   int shmid;
   char *shared_memory;
   char buff[^100];

   shmid = shmget(2345, 1024, 0666 | IPC_CREAT);
   printf("Shared memory key: %d\n", shmid);

   shared_memory = (char*) shmat(shmid, NULL, 0);
   printf("Attached at: %p\n", shared_memory);

   printf("Enter data to write to shared memory:\n");
   read(0, buff, sizeof(buff));
   strcpy(shared_memory, buff);

   printf("You wrote: %s", shared_memory);
   return 0;
}
```

Output Example:

```
Shared memory key: 12345
Attached at: 0x7f9b8c000000
Enter data to write to shared memory:
Hello Shared Memory!
You wrote: Hello Shared Memory!
```

(b) Pipes

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

int main() {
   int fd[^2], n;
   char buff[^100];
   pipe(fd);
   pid_t p = fork();

   if (p > 0) {
      close(fd[^0]);
      printf("Parent: Sending message to child...\n");
      write(fd[^1], "My name is ANKIT KUMAR\n", 23);
      close(fd[^1]);
      wait(NULL);
   } else {
      close(fd[^1]);
      printf("Child: Received message -\n");
      n = read(fd[^0], buff, sizeof(buff));
      write(1, buff, n);
      close(fd[^0]);
   }
   return 0;
}
```

Output:

```
Parent: Sending message to child...
Child: Received message -
My name is ANKIT KUMAR
```

(c) Message Queue

```c
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/msg.h>

struct msg_buffer {
    long msg_type;
    char msg[^100];
};

int main() {
    int msgid = msgget(ftok("progfile", 65), 0666 | IPC_CREAT);
    struct msg_buffer message = {1};

    printf("Write Message: ");
    fgets(message.msg, sizeof(message.msg), stdin);

    msgsnd(msgid, &message, sizeof(message), 0);
    printf("Sent: %s", message.msg);

    msgrcv(msgid, &message, sizeof(message), 1, 0);
    printf("Received: %s", message.msg);

    msgctl(msgid, IPC_RMID, NULL);
    return 0;
}
```

Output Example:

```
Write Message: ANKIT
Sent: ANKIT
Received: ANKIT
```

EXPERIMENT NO. 5

AIM: Write C program to simulate solutions to classical process synchronization problems: Dining Philosopher.

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

#define NUM_PHILOSOPHERS 5
#define NUM_CHOPSTICKS 5

void* dine(void* num);
pthread_t philosopher[NUM_PHILOSOPHERS];
pthread_mutex_t chopstick[NUM_CHOPSTICKS];

int main() {
   int i, status_message;
   void *msg;

   for (i = 0; i < NUM_CHOPSTICKS; i++) {
      status_message = pthread_mutex_init(&chopstick[i], NULL);
      if (status_message != 0) {
         printf("\nMutex initialization failed\n");
         exit(1);
      }
   }

   for (i = 0; i < NUM_PHILOSOPHERS; i++) {
      int* id = malloc(sizeof(int));
      *id = i;
      status_message = pthread_create(&philosopher[i], NULL, dine, id);
```

```
        if (status_message != 0) {
            printf("\nThread creation error\n");
            exit(1);
        }
    }

    for (i = 0; i < NUM_PHILOSOPHERS; i++) {
        status_message = pthread_join(philosopher[i], &msg);
        if (status_message != 0) {
            printf("\nThread join failed\n");
            exit(1);
        }
    }

    for (i = 0; i < NUM_CHOPSTICKS; i++) {
        status_message = pthread_mutex_destroy(&chopstick[i]);
        if (status_message != 0) {
            printf("\nMutex Destroy failed\n");
            exit(1);
        }
    }
    return 0;
}

void* dine(void* num) {
    int n = *(int*)num;
    free(num);

    printf("\nPhilosopher %d is thinking", n);

    pthread_mutex_lock(&chopstick[n]);
    pthread_mutex_lock(&chopstick[(n + 1) % NUM_CHOPSTICKS]);

    printf("\nPhilosopher %d is eating", n);
```

```
    sleep(3);

    pthread_mutex_unlock(&chopstick[n]);
    pthread_mutex_unlock(&chopstick[(n + 1) % NUM_CHOPSTICKS]);

    printf("\nPhilosopher %d finished eating", n);
    return NULL;
}
```

Output Example:

```
Philosopher 0 is thinking
Philosopher 1 is thinking
Philosopher 2 is thinking
Philosopher 3 is thinking
Philosopher 4 is thinking
Philosopher 0 is eating
Philosopher 1 is eating
Philosopher 0 finished eating
Philosopher 2 is eating
Philosopher 1 finished eating
Philosopher 3 is eating
Philosopher 2 finished eating
Philosopher 4 is eating
Philosopher 3 finished eating
Philosopher 4 finished eating
```

EXPERIMENT NO. 6

AIM: Write C program to simulate Banker's Algorithm for deadlock avoidance.

```
#include <stdio.h>

#define true 1
#define false 0

int available[^10], allocation[^10][^10], max[^10][^10], need[^10][^10], work[^10], finish[^10], maxres[^10], safe[^10], m, n;

int find() {
   int i, j;
   for (i = 0; i < n; i++) {
      if (finish[i] == false) {
         for (j = 0; j < m; j++)
            if (need[i][j] > work[j])
               break;
         if (j == m) {
            finish[i] = true;
            return i;
         }
      }
   }
   return -1;
}

int issafe() {
   int i = 0, j, k = 0, cnt = n;
   for (j = 0; j < m; j++)
      work[j] = available[j];
   for (j = 0; j < n; j++)
      finish[j] = false;

   while (cnt > 0) {
      i = find();
      if (i == -1) {
         printf("\nThe system is in unsafe state\n");
         return 0;
      }
      for (j = 0; j < m; j++)
         work[j] += allocation[i][j];
      safe[k++] = i;
      cnt--;
```

```
    }

    printf("\nThe system is in safe state, safe sequence: ");
    for (i = 0; i < n; i++)
        printf("P%d ", safe[i] + 1);
    printf("\n");
    return 1;
}

int main() {
    int i, j, sum;

    printf("\nEnter the number of processes and the number of resources:\n");
    scanf("%d%d", &n, &m);

    printf("\nEnter maximum instances of resources\n");
    for (j = 0; j < m; j++) {
        scanf("%d", &maxres[j]);
        available[j] = maxres[j];
    }

    printf("\nEnter the Allocated Matrix:\n");
    for (i = 0; i < n; i++)
        for (j = 0; j < m; j++)
            scanf("%d", &allocation[i][j]);

    printf("\nEnter the Max Matrix:\n");
    for (i = 0; i < n; i++) {
        for (j = 0; j < m; j++) {
            scanf("%d", &max[i][j]);
            need[i][j] = max[i][j] - allocation[i][j];
        }
    }

    printf("\nThe Need Matrix is:\n");
    for (i = 0; i < n; i++) {
        for (j = 0; j < m; j++)
            printf("%d ", need[i][j]);
        printf("\n");
    }

    for (j = 0; j < m; j++) {
        sum = 0;
        for (i = 0; i < n; i++)
            sum += allocation[i][j];
```

```
        available[j] -= sum;
    }

    issafe();

    return 0;
}
```

Output Example:

```
Enter the number of processes and the number of resources:
5 3

Enter maximum instances of resources
10 5 7

Enter the Allocated Matrix:
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2

Enter the Max Matrix:
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3

The Need Matrix is:
7 4 3
1 2 2
6 0 0
0 1 1
4 3 1

The system is in safe state, safe sequence: P1 P3 P4 P5 P2
```

EXPERIMENT NO. 7

AIM: Write C program to simulate page replacement algorithms: FIFO and LRU.

(a) FIFO Page Replacement

```
#include <stdio.h>

int main() {
   int n, a[^50], f, frame[^50], j = 0, count = 0, hit;

   printf("Enter reference string length: ");
   scanf("%d", &n);

   printf("Enter reference string: ");
   for (int i = 0; i < n; i++)
      scanf("%d", &a[i]);

   printf("Enter no. of frames: ");
   scanf("%d", &f);

   for (int i = 0; i < f; i++)
      frame[i] = -1;

   printf("\nReference\tFrames\n");
   for (int i = 0; i < n; i++) {
      printf("%d\t\t", a[i]);
      hit = 0;
      for (int k = 0; k < f; k++)
         if (frame[k] == a[i]) {
            hit = 1;
            break;
         }
      if (!hit) {
         frame[j] = a[i];
         j = (j + 1) % f;
         count++;
      }
      for (int k = 0; k < f; k++)
         printf("%d\t", frame[k] == -1 ? '-' : frame[k]);
      if (hit)
         printf("Hit");
```

```
        printf("\n");
    }

    printf("\nTotal Page Faults = %d\n", count);
    return 0;
}
```

Output Example:

```
Enter reference string length: 12
Enter reference string: 1 2 3 4 1 2 5 1 2 3 4 5
Enter no. of frames: 3

Reference      Frames
1              1     -     -
2              1     2     -
3              1     2     3
4              4     2     3
1              4     1     3
2              4     1     2
5              5     1     2
1              5     1     2     Hit
2              5     1     2     Hit
3              3     1     2
4              3     4     2
5              3     4     5

Total Page Faults = 9
```

(b) LRU Page Replacement

```
#include <stdio.h>

void print(int f, int fr[]) {
    for (int i = 0; i < f; i++)
        printf("%d\t", fr[i]);
    printf("\n");
}

int main() {
```

```
int n, p[^50], f, fr[^10], freq[^10] = {0}, count = 0, used = 0, rep, min;

printf("Enter no. of pages: ");
scanf("%d", &n);

printf("Enter page refs: ");
for (int i = 0; i < n; i++)
   scanf("%d", &p[i]);

printf("Enter no. of frames: ");
scanf("%d", &f);

for (int i = 0; i < f; i++)
   fr[i] = -1;

printf("Page\tFrames\n");
for (int i = 0; i < n; i++) {
   printf("%d\t", p[i]);
   int found = 0;
   for (int j = 0; j < f; j++) {
      if (fr[j] == p[i]) {
         freq[j]++;
         found = 1;
         printf("No replace\n");
         break;
      }
   }
   if (!found) {
      if (used < f) {
         fr[used] = p[i];
         freq[used++] = 1;
         count++;
      } else {
         rep = 0;
         min = freq[^0];
         for (int j = 1; j < f; j++)
            if (freq[j] < min) {
```

```
                min = freq[j];
                rep = j;
            }
        fr[rep] = p[i];
        freq[rep] = 1;
        count++;
      }
      print(f, fr);
    }
  }

  printf("\nTotal Page Faults = %d\n", count);
  return 0;
}
```

Output Example:

```
Enter no. of pages: 12
Enter page refs: 1 2 3 4 1 2 5 1 2 3 4 5
Enter no. of frames: 3
Page    Frames
1       1       -       -
2       1       2       -
3       1       2       3
4       4       2       3
1       No replace
2       No replace
5       5       2       3
1       5       1       3
2       5       1       2
3       3       1       2
4       3       4       2
5       3       4       5

Total Page Faults = 9
```

EXPERIMENT NO. 8

Aim: Write C program to simulate implementation of Disk Scheduling algorithms: FCFS , SSTF

(i)FCFS Disk Scheduling Algorithm Program in C

```
#include<math.h>
#include<stdio.h>
#include<stdlib.h>
int main()
{
   int i,n,req[50],mov=0,cp;
   printf("enter the current position\n");
   scanf("%d",&cp);
   printf("enter the number of requests\n");
   scanf("%d",&n);
   printf("enter the request order\n");
   for(i=0;i<n;i++)
   {
      scanf("%d",&req[i]);
   }
   mov=mov+abs(cp-req[0]); // abs is used to calculate the absolute value
   printf("%d -> %d",cp,req[0]);
   for(i=1;i<n;i++)
   {
      mov=mov+abs(req[i]-req[i-1]);
      printf(" -> %d",req[i]);
   }
   printf("\n");
   printf("total head movement = %d\n",mov);
}
```

Output:

```
enter the current position:45
enter the number of requests:5
enter the request order:20 89 34 56 19
45 -> 20 -> 89 -> 34 -> 56 -> 19
total head movement = 208
```

(ii)SSTF Disk Scheduling Algorithm Program in C

```
#include<math.h>
#include<stdio.h>
#include<stdlib.h>
int main()
{
   int i,n,k,req[50],mov=0,cp,index[50],min,a[50],j=0,mini,cp1;
   printf("enter the current position\n");
   scanf("%d",&cp);
   printf("enter the number of requests\n");
   scanf("%d",&n);
   cp1=cp;
   printf("enter the request order\n");
   for(i=0;i<n;i++)
   {
      scanf("%d",&req[i]);
   }
   for(k=0;k<n;k++)
   {
   for(i=0;i<n;i++)
   {
      index[i]=abs(cp-req[i]); // calculate distance of each request from current position
   }
   // to find the nearest request
```

```
   min=index[0];
   mini=0;
   for(i=1;i<n;i++)
   {
      if(min>index[i])
      {
         min=index[i];
         mini=i;
      }
   }
   a[j]=req[mini];
   j++;
   cp=req[mini]; // change the current position value to next request
   req[mini]=999;
   } // the request that is processed its value is changed so that it is not processed again
   printf("Sequence is : ");
   printf("%d",cp1);
   mov=mov+abs(cp1-a[0]);    // head movement
   printf(" -> %d",a[0]);
   for(i=1;i<n;i++)
   {
      mov=mov+abs(a[i]-a[i-1]); ///head movement
      printf(" -> %d",a[i]);
   }
   printf("\n");
   printf("total head movement = %d\n",mov);
}
```

Output:

```
enter the current position: 45

enter the number of requests: 5

enter the request order: 20 89 34 56 19

Sequence is : 45 -> 34 -> 20 -> 19 -> 56 -> 89

total head movement = 96
```