OS PRACTICAL EXAMINATION

List of Expt for OR-PR Exam		
		Implement Any One
1	1	SHELL PROGRAM FOR OPENDIR(), READDIR() FUNCTION
	2	Implement all file allocation strategies: Sequential
	3	Implement all File Organization Techniques:a) Single level directory
2	1	SHELL PROGRAM TO SIMULATE THE FORK( ) AND GETPID( ) SYSTEM CALLS
	2	Implement all file allocation strategies: Indexed
	3	Implement all File Organization Techniques: Single level directory
3	1	Implement Semaphores for Producer Consumer Problem.
	2	SHELL PROGRAM TO DEMONSTRATE EXECLP() FUNCTION
	3	Implement Shared memory and IPC.
4	1	Implement Bankers Algorithm for Dead Lock Avoidance.
	2	Implement all page replacement algorithms: FIFO
	3	SHELL PROGRAM TO DEMONSTRATE SLEEP() FUNCTION
5	1	Implement an Algorithm for Dead Lock Detection.
	2	Implement Paging Technique of memory management.
	3	Implement all File Organization Techniques: Hierarchical
6	1	Write a program to perform first come first serve scheduling algorithm
	2	Implement all file allocation strategies: Linked
	3	Implement Semaphores for Producer Consumer Problem.
7	1	Write a program to perform SHORTEST JOB FIRST SCHEDULING ALGORITHM
	2	Implement all page replacement algorithms:LRU
	3	UNIX commands
8	1	PRIORITY SCHEDULING ALGORITHM
	2	Implement all File Organization Techniques: Hierarchical
	3	Implement all page replacement algorithms: LFU
9	1	ROUND ROBIN SCHEDULING
	2	Implement all page replacement algorithms: LFU
	3	Implement Bankers Algorithm for Dead Lock Avoidance.
10	1	UNIX commands
	2	Implement all File Organization Techniques: Two level
	3	Implement Shared memory and IPC.

The list contains the source code for all ten practicals, indexed by number. Note that 3.3 and 10.3 are the same, so you only need to prepare nine practicals.

1.2 Implement All File Allocation Stratergies :Sequential

```
#include <stdio.h>

int main() {
    int n, i, j, b[20], sb[20], t[20], x, c[20][20];
printf("Enter no. of files:");
scanf("%d", &n);
    for (i = 0; i< n; i++) {
printf("Enter no. of blocks occupied by file %d: ", i + 1);
scanf("%d", &b[i]);
printf("Enter the starting block of file %d: ", i + 1);
scanf("%d", &sb[i]);
        t[i] = sb[i];

        for (j = 0; j < b[i]; j++) {
            c[i][j] = sb[i]++;
        }
    }
printf("Filename Start blockLength\n");
    for (i = 0; i< n; i++) {
printf("%d\t\t %d \t\t\t%d\n", i + 1, t[i], b[i]);
    }
printf("Blocks occupied are:\n");
    for (i = 0; i< n; i++) {
printf("File no %d: ", i + 1);
        for (j = 0; j < b[i]; j++) {
printf("\t%d", c[i][j]);
        }
printf("\n");
    }
    return 0;
}
```

OUTPUT:

```
~/Operating-System-LAB-Practicals$ ./sequential
Enter no. of files:2
Enter no. of blocks occupied by file 1: 3
Enter the starting block of file 1: 4
Enter no. of blocks occupied by file 2: 2
Enter the starting block of file 2: 8
Filename Start blockLength
1       4               3
2       8               2
Blocks occupied are:
File no 1:      4       5       6
File no 2:      8       9
~/Operating-System-LAB-Practicals$
```

2.1 Shell Program To Simulate The Fork( ) And Getpid( ) System Calls

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

int main() {
    pid_t pid;

    printf("Before fork - Process ID: %d\n", getpid());

    // Simulate fork
    pid = fork();

    if (pid == -1) {
        // Fork failed
        perror("fork");
        return 1;
    } else if (pid == 0) {
        // Child process
        printf("After fork - Child Process ID: %d\n", getpid());
    } else {
        // Parent process
        printf("After fork - Parent Process ID: %d\n",
getpid());
    }

    return 0;
}
```

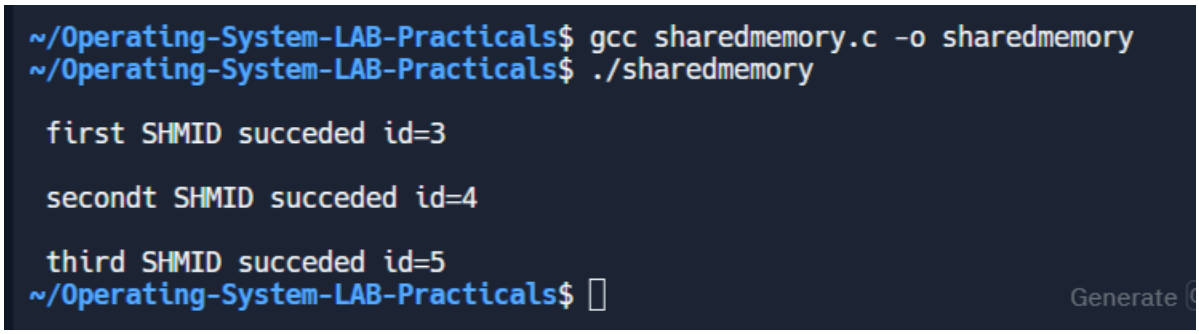
Output:

```
svkm@svkm-VirtualBox:~$ cd shell
svkm@svkm-VirtualBox:~/shell$ gcc getpid.c
svkm@svkm-VirtualBox:~/shell$ ./a.out
Before fork - Process ID: 2349
After fork- Parent Process ID: 2349
svkm@svkm-VirtualBox:~/shell$ After fork- Child Process ID: 2350
```

3.3 Implement Shared Memory And IPC

```
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/types.h>
#include <stdio.h>

int main() {
  int shmid;
key_t key = 0 * 10;
shmid = shmget(key, 100, IPC_CREAT | 0666);
  if (shmid< 0)
printf("\nfirst SHMID failed\n");
  else
printf("\n first SHMID succeded id=%d\n", shmid);
shmid = shmget(key, 101, IPC_CREAT | 0666);
  if (shmid< 0)
printf("\nsecond SHMID failed\n");
  else
printf("\n secondt SHMID succeded id=%d\n", shmid);
shmid = shmget(key, 90, IPC_CREAT | 0666);
  if (shmid< 0)
printf("\nthird SHMID failed\n");
  else
printf("\n third SHMID succeded id=%d\n", shmid);
}
```

OUTPUT:



4.3 Shell Program To Demonstrate Sleep() Function

```
#include<stdio.h>
#include<unistd.h>
#include<sys/types.h>
int main()
{
    int p;
    p=fork();
    if (p=0)
    {
            printf("\n I am a child processed",getpid());
            sleep(15);
            printf("\n I am a parent of child processed",
getpid());
    }
    else
    {
            printf("\n I am a parent of child%d", getpid());
            printf("\n I am a parent's parent%d", getpid());
   }
   return 0;
}
```

Output:

```
I am a parent of child11166

I am a parent of child11167
I am a parent's parent11166 I am a parent's parent11167

=== Code Execution Successful ===
```

5.1 Implement Algorithm For Deadlock Detection

```
#include <stdio.h>
int main() {
  int found, flag, l, p[10][10], tp, tr, c[10][10];
  int i, j, k = 1, m[10], r[10], a[10], temp[10], sum = 0;
printf("Enter total no of processes: ");
scanf("%d", &tp);
printf("Enter total no of resources: ");
scanf("%d", &tr);
printf("Enter claim (Max. Need) matrix\n");
  for (i = 1; i<= tp; i++) {
printf("process %d:\n", i);
    for (j = 1; j <= tr; j++)
scanf("%d", &c[i][j]);
  }
printf("Enter allocation matrix\n");
  for (i = 1; i<= tp; i++) {
printf("process %d:\n", i);
    for (j = 1; j <= tr; j++)
scanf("%d", &p[i][j]);
  }
printf("Enter resource vector (Total resources):\n");
  for (i = 1; i<= tr; i++)
scanf("%d", &r[i]);
printf("Enter availability vector (available resources):\n");
  for (i = 1; i<= tr; i++) {
scanf("%d", &a[i]);
    temp[i] = a[i];
  }
  for (i = 1; i<= tp; i++) {
    sum = 0;
    for (j = 1; j <= tr; j++)
      sum += p[i][j];
    if (sum == 0) {
      m[k] = i;
      k++;
    }
  }
  for (i = 1; i<= tp; i++) {
    for (l = 1; l < k; l++)
      if (i != m[l]) {
        flag = 1;
        for (j = 1; j <= tr; j++)
          if (c[i][j] < temp[j]) {
            flag = 0;
            break;
          }
      }
    if (flag == 1) {
      m[k] = i;
      k++;
```

```
      for (j = 1; j <= tr; j++)
        temp[j] += p[i][j];
    }
  }
printf("deadlock causing processes are: ");
  for (j = 1; j <= tp; j++) {
    found = 0;
    for (i = 1; i< k; i++) {
      if (j == m[i])
        found = 1;
    }
    if (found == 0)
printf("%d\t", j);
  }
}
```

OUTPUT:

```
~/Operating-System-LAB-Practicals$ ./deadlockdetect
Enter total no of processes: 4
Enter total no of resources: 5
Enter claim (Max. Need) matrix
process 1:
0 1 0 0 1
process 2:
0 0 1 0 1
process 3:
0 0 0 0 1
process 4:
1 0 1 0 1
Enter allocation matrix
process 1:
1 0 1 1 0
process 2:
1 1 0 0 0
process 3:
0 0 0 1 0
process 4:
0 0 0 0 0
Enter resource vector (Total resources):
2 1 1 2 1
Enter availability vector (available resources):
0 0 0 0 1
~/Operating-System-LAB-Practicals$
```

6.2 Implement All File Allocation Stratergies : Linked

```
#include <stdio.h>

struct file {
    char fname[10];
    int start, size, block[10];
} f[10];

int main() {
    int i, j, n;
printf("Enter no. of files:");
scanf("%d", &n);
for(i = 0; i< n; i++) {
printf("Enter file name:");
scanf("%s", &f[i].fname);

printf("Enter starting block:");
scanf("%d", &f[i].start);
        f[i].block[0] = f[i].start;

printf("Enter no.of blocks:");
scanf("%d", &f[i].size);

printf("Enter block numbers:");
for(j = 1; j <= f[i].size; j++) {
scanf("%d", &f[i].block[j]);
        }
    }
printf("File\tstart\tsize\tblock\n");
for(i = 0; i< n; i++) {
printf("%s\t%d\t%d\t", f[i].fname, f[i].start, f[i].size);
for(j = 0; j < f[i].size; j++) {
printf("%d--->", f[i].block[j]);
        }
printf("%d", f[i].block[j]);
printf("\n");
    }
    return 0;
}
```

OUTPUT:

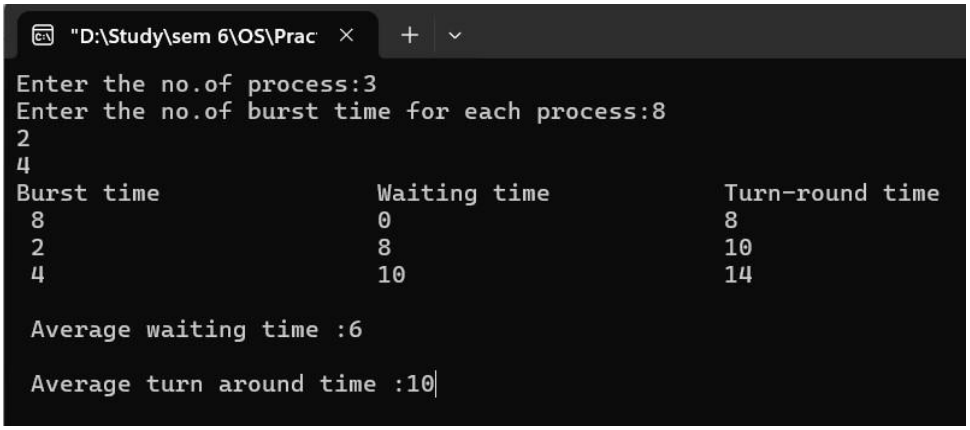
```
/tmp/GNR4uuG4Qv.o
Enter no. of files:2
Enter file name:3
Enter starting block:1
Enter no.of blocks:3
Enter block numbers:2 4 5
Enter file name:3
Enter starting block:2
Enter no.of blocks:3
Enter block numbers:4 5 2
File    start   size    block
3   1   3   1--->2--->4--->5
3   2   3   2--->4--->5--->2


=== Code Execution Successful ===
```

7.1 Write a program to perform SHORTEST JOB FIRST SCHEDULING ALGORITHM and compute the average waiting time and average turnaround time.

```
#include <stdio.h>

int main()
{
   int bt[20], p, wt = 0, tat, i, j, twt = 0, ttat = 0, temp;

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

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

   // Sorting burst time in ascending order
   for (i = 0; i < p - 1; i++)
   {
      for (j = i + 1; j < p; j++)
      {
         if (bt[i] > bt[j])
         {
            temp = bt[i];
            bt[i] = bt[j];
            bt[j] = temp;
         }
      }
   }

   printf("Burst time\tWaiting time\tTurn-around time\n");

   for (i = 0; i < p; i++)
   {
      tat = bt[i] + wt;
      twt += wt;
      ttat += tat;
      printf("%-12d\t%-12d\t%-15d\n", bt[i], wt, tat);  // Adjusted spacing for alignment
      wt += bt[i];
   }
```

```
    printf("\nAverage waiting time: %.2f", (float)twt / p);
    printf("\nAverage turn-around time: %.2f", (float)ttat / p);

    return 0;
}
```

Output:



8.1 Write a program to create and execute PRIORITY SCHEDULING ALGORITHM using c program.

```
#include <stdio.h>

void main() {
   int i, j, pno[10], prior[10], bt[10], n, wt[10], tt[10], w1 = 0, t1 = 0, s;
   float aw, at;

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

   for (i = 0; i < n; i++) {
      printf("Process %d:\n", i + 1);
      printf("Enter the burst time of process: ");
      scanf("%d", &bt[i]);
      printf("Enter the priority of process %d: ", i + 1);
      scanf("%d", &prior[i]);
      pno[i] = i + 1;
   }

   // Sorting based on priority (lower number indicates higher priority)
   for (i = 0; i < n - 1; i++) {
      for (j = i + 1; j < n; j++) {
         if (prior[i] > prior[j]) {
            // Swapping priority
            s = prior[i];
            prior[i] = prior[j];
            prior[j] = s;
```

```
            // Swapping burst time
            s = bt[i];
            bt[i] = bt[j];
            bt[j] = s;

            // Swapping process number
            s = pno[i];
            pno[i] = pno[j];
            pno[j] = s;
        }
    }
}

// Calculate waiting time and turnaround time
wt[0] = 0;
tt[0] = bt[0];
w1 = wt[0];
t1 = tt[0];

for (i = 1; i < n; i++) {
    wt[i] = wt[i - 1] + bt[i - 1];
    tt[i] = wt[i] + bt[i];
    w1 += wt[i];
    t1 += tt[i];
}

aw = (float)w1 / n;
at = (float)t1 / n;
```

```
    // Printing the results
    printf("\nJob\tBT\tWT\tTAT\tPriority\n");
    for (i = 0; i < n; i++) {
        printf("%d\t%d\t%d\t%d\t%d\n", pno[i], bt[i], wt[i], tt[i], prior[i]);
    }
    printf("\nAverage Waiting Time: %.2f", aw);
    printf("\nAverage Turnaround Time: %.2f\n", at);
}
```

OUTPUT:

```
enter the number of processes:4
The process 1:
Enter the burst time of processes:24
Enter the priority of processes 1:3
The process 2:
Enter the burst time of processes:15
Enter the priority of processes 2:2
The process 3:
Enter the burst time of processes:32
Enter the priority of processes 3:5
The process 4:
Enter the burst time of processes:12
Enter the priority of processes 4:4

 job          bt       wt       tat      prior
  2           15       0        15         2
  1           24       15       39         3
  4           12       39       51         4
  3           32       51       83         5
aw=26.250000      at=47.000000
```

9.1 Write a program to implement ROUND ROBIN SCHEDULING ALGORITHMS using C.

```
#include<stdio.h>
void main() {
   int b[10], i, j = 1, n, temp, burst[10], wait[10], turn[10], p[10], a = 1, q, tat[10], t1 = 0;
   float t = 0, w = 0;
   printf("Enter the number of processes & time quantum: ");
   scanf("%d%d", &n, &q);
   printf("Enter burst time: ");
   for (i = 1; i <= n; i++)
      scanf("%d", &burst[i]);
   burst[0] = 0;
   b[0] = 0;
   tat[0] = 0;
   p[0] = 0;
   printf("\n\n\t\t Gantt chart\n");
   printf("  \n");
   for (i = 1; i <= n; i++)
      b[i] = burst[i];
   for (i = 1; i <= n; i++) {
      if (b[i] > 0) {
         a = 1;
         printf("P%d\t|", i);
         if (b[i] >= q) {
            t1 = t1 + q;
            p[j] = t1;
            j++;
         } else if (b[i] < q) {
            t1 = t1 + b[i];
```

```
            p[j] = t1;
            j++;
        }
        b[i] = b[i] - q;

        if (b[i] <= 0)
            tat[i] = t1;
    } else
        a++;
    if (a == n + 1)
        break;

    if (i == n)
        i = 0;
}

printf("\n  \n");

for (i = 0; i < j; i++)
    printf("%d\t", p[i]);

for (i = 1; i <= n; i++) {
    t = t + tat[i];
    w = w + tat[i] - burst[i];
}

w = w / n;
t = t / n;
```

```
    printf("\nThe average waiting time is %.2f", w);
    printf("\nThe average turn around time is %.2f", t);
}
```

Output:

```
Enter the number of processes & time quantum: 5
3
Enter burst time: 4
7
5
3
2


         Gantt chart

P1  |P2 |P3 |P4 |P5 |P1 |P2 |P3 |P2 |

0   3   6   9   12  14  15  18  20  21
The average waiting time is 12.20
The average turn around time is 16.40
```

10.3 Implement Shared Memory And IPC

```
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/types.h>
#include <stdio.h>

int main() {
  int shmid;
key_t key = 0 * 10;
shmid = shmget(key, 100, IPC_CREAT | 0666);
  if (shmid< 0)
printf("\nfirst SHMID failed\n");
  else
printf("\n first SHMID succeded id=%d\n", shmid);
shmid = shmget(key, 101, IPC_CREAT | 0666);
  if (shmid< 0)
printf("\nsecond SHMID failed\n");
  else
printf("\n secondt SHMID succeded id=%d\n", shmid);
shmid = shmget(key, 90, IPC_CREAT | 0666);
  if (shmid< 0)
printf("\nthird SHMID failed\n");
  else
printf("\n third SHMID succeded id=%d\n", shmid);
}
```

OUTPUT: