Operating System Lab Observation

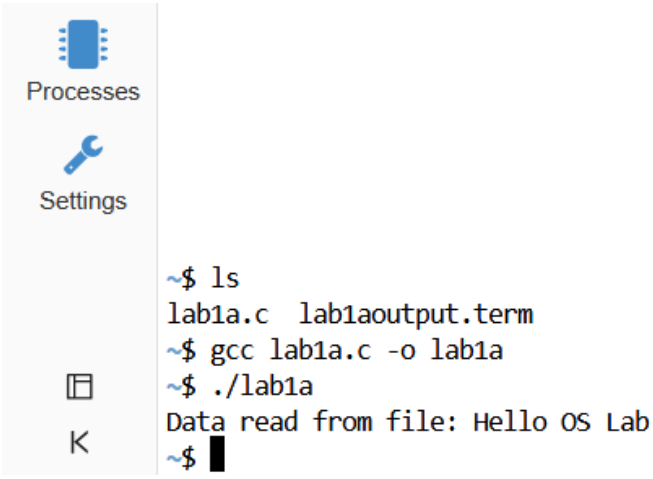
1a

Program:

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

int main()
{
   int fd;
   char buffer[50];

   /* open() – create and open a file */
   fd = open("sample.txt", O_CREAT | O_RDWR, 0644);

   if (fd < 0)
   {
      printf("File opening failed\n");
      return 1;
   }

   /* write() – write data into file */
   write(fd, "Hello OS Lab", 12);

   /* move file pointer to beginning */
   lseek(fd, 0, SEEK_SET);
```

```
    /* read() – read data from file */
    read(fd, buffer, 12);

    /* display data */
    printf("Data read from file: %s\n", buffer);

    /* close() – close the file */
    close(fd);

    return 0;
}
```

Output:

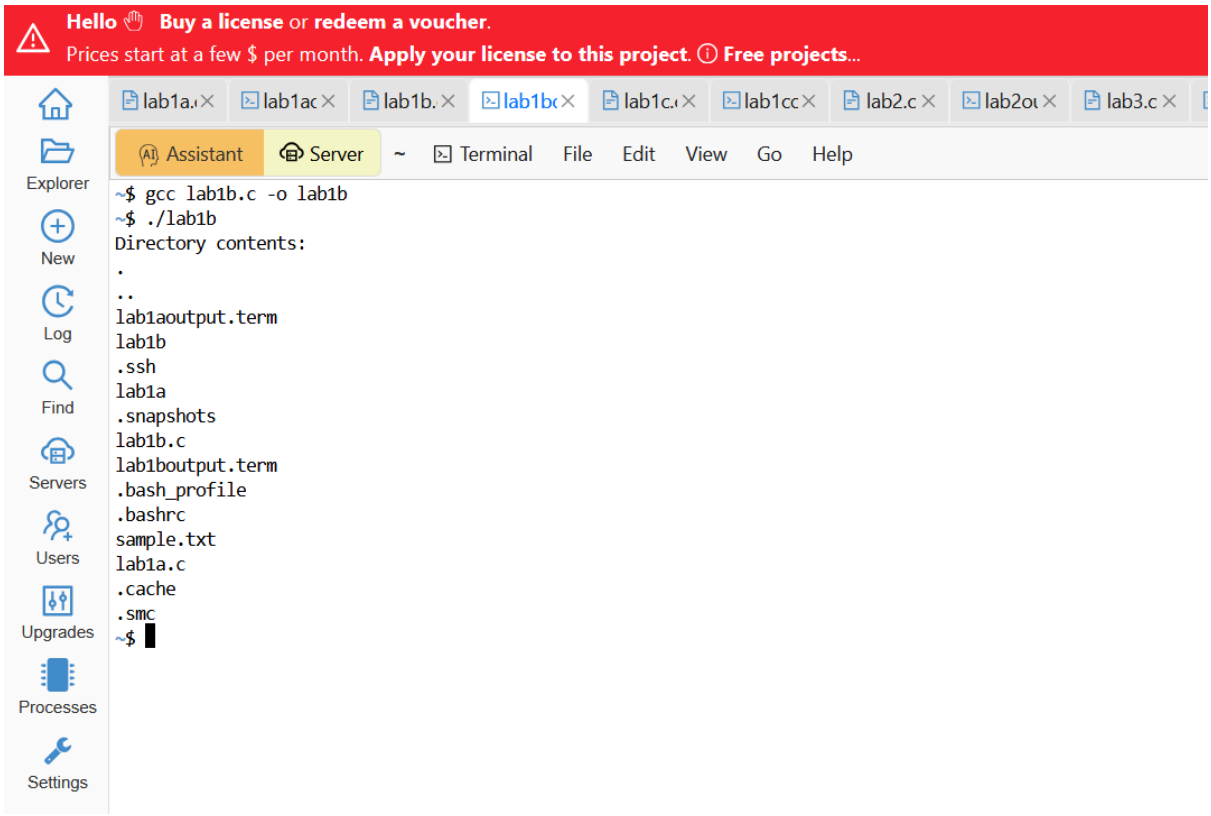


1b:

Program:

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

int main()
{
   DIR *d;
   struct dirent *entry;

   /* open current directory */
   d = opendir(".");

   if (d == NULL)
   {
      printf("Directory cannot be opened\n");
      return 1;
   }
   /* read directory entries */
   printf("Directory contents:\n");
   while ((entry = readdir(d)) != NULL)
   {
      printf("%s\n", entry->d_name);
   }
   /* close directory */
   closedir(d);
   return 0;
}
```

Output:



1c:

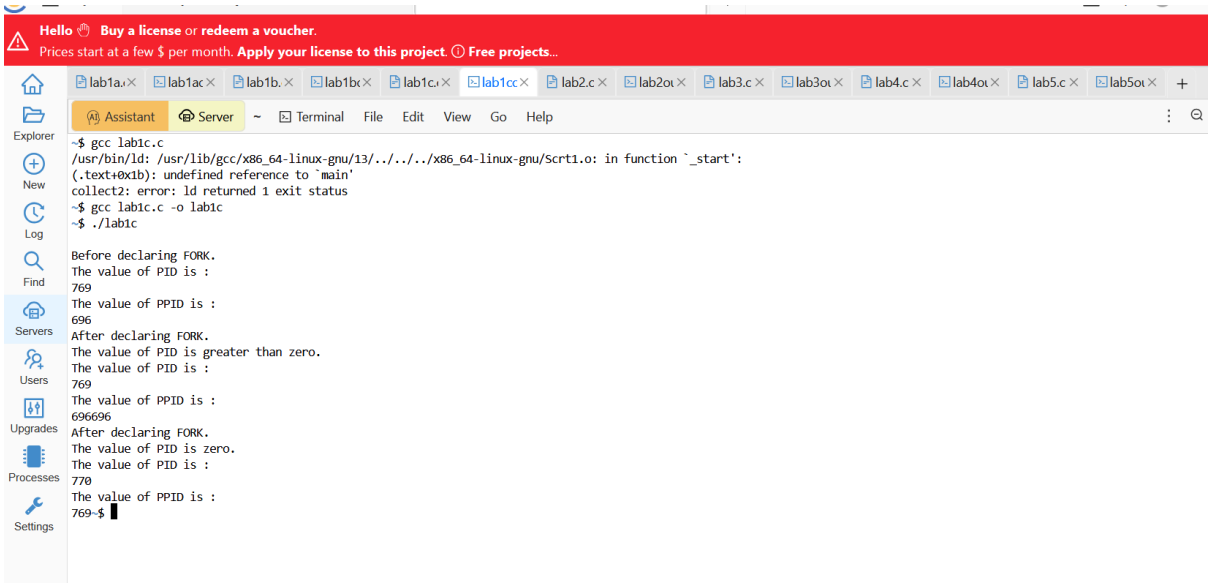
Program:

```
#include<stdio.h>

#include<unistd.h>

#include<sys/types.h>

void main()

{

printf("\nBefore declaring FORK.");

printf("\nThe value of PID is : ");

printf("\n%d",getpid());

printf("\nThe value of PPID is : ");

printf("\n%d",getppid());
```

```
pid_t pid=fork();//Used to create child process
if(pid==0)//Child Process
{
printf("\nAfter declaring FORK.");
printf("\nThe value of PID is zero.");
printf("\nThe value of PID is : ");
printf("\n%d",getpid());
printf("\nThe value of PPID is : ");
printf("\n%d",getppid());
}
else if(pid>0)//Parent Process
{
printf("\nAfter declaring FORK.");
printf("\nThe value of PID is greater than zero.");
printf("\nThe value of PID is : ");
printf("\n%d",getpid());
printf("\nThe value of PPID is : ");
printf("\n%d",getppid());
}
else//Error
{
printf("\nAfter declaring FORK.");
printf("\nThe value of PID is less than zero.");
printf("\nThe value of PID is : ");
printf("\n%d",getpid());
printf("\nThe value of PPID is : ");
printf("%d",getppid());
}
```

}

Output:



Lab2:

Program:

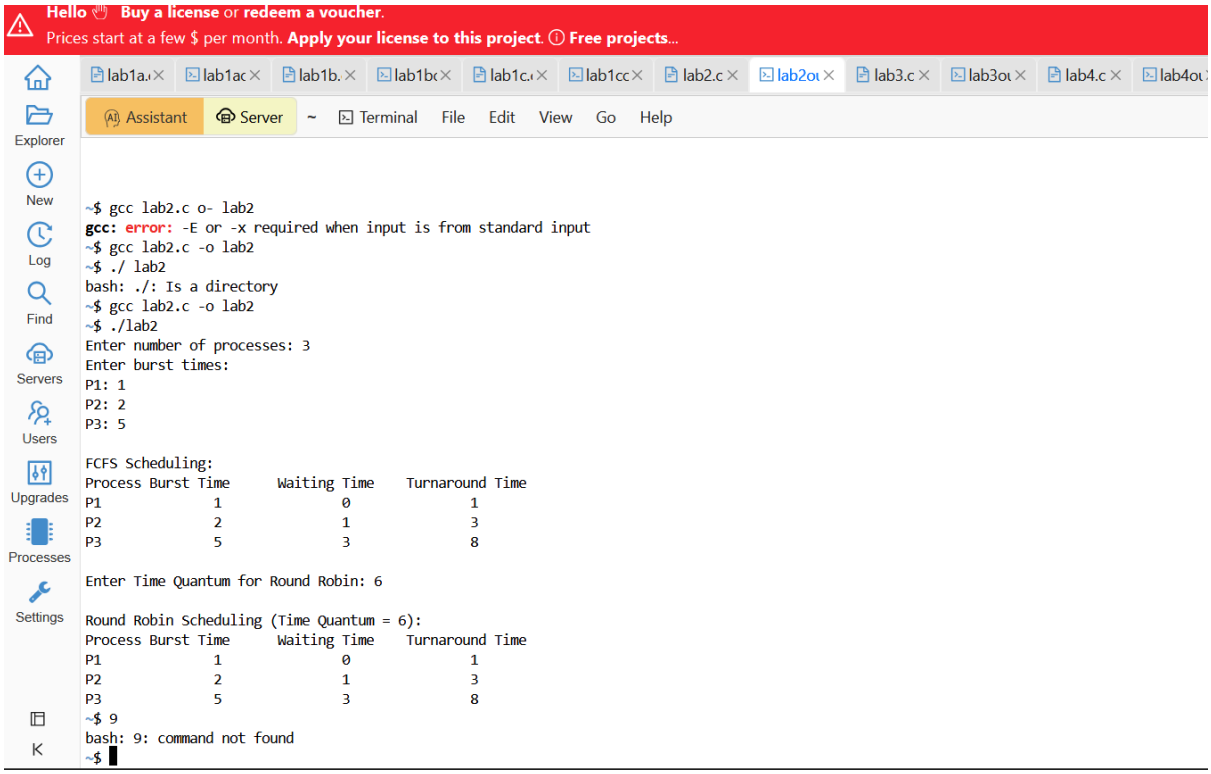
```
#include <stdio.h>

// FCFS Scheduling

void fcfs(int n, int bt[]) {

int wt[10], tat[10];

wt[0] = 0;

for(int i = 1; i < n; i++)

wt[i] = wt[i-1] + bt[i-1];

printf("\nFCFS Scheduling:\n");

printf("Process\tBurst Time\tWaiting Time\tTurnaround Time\n");

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

```
tat[i] = wt[i] + bt[i];
printf("P%d\t\t%d\t\t%d\t\t%d\n", i+1, bt[i], wt[i], tat[i]);
}
}
// Round Robin Scheduling
void roundRobin(int n, int bt[], int tq) {
int rem_bt[10], wt[10], tat[10];
int t = 0;
for(int i = 0; i < n; i++) {
rem_bt[i] = bt[i];
wt[i] = 0;
}
while(1) {
int done = 1;
for(int i = 0; i < n; i++) {
if(rem_bt[i] > 0) {
done = 0;
if(rem_bt[i] > tq) {
t += tq;
                rem_bt[i] -= tq;
            } else {
                t += rem_bt[i];
                wt[i] = t - bt[i];
                rem_bt[i] = 0;
            }
        }
    }
    if(done == 1)
```

```
            break;
        }

        printf("\nRound Robin Scheduling (Time Quantum = %d):\n", tq);
        printf("Process\tBurst Time\tWaiting Time\tTurnaround Time\n");

        for(int i = 0; i < n; i++) {
            tat[i] = bt[i] + wt[i];
            printf("P%d\t\t%d\t\t%d\t\t%d\n", i+1, bt[i], wt[i], tat[i]);
        }
    }

    int main() {
        int n, tq;
        int bt[10];

        printf("Enter number of processes: ");
    scanf("%d", &n);
    printf("Enter burst times:\n");
    for(int i = 0; i < n; i++) {
    printf("P%d: ", i+1);
    scanf("%d", &bt[i]);
    }
    fcfs(n, bt);
    printf("\nEnter Time Quantum for Round Robin: ");
    scanf("%d", &tq);
    roundRobin(n, bt, tq);
    return 0;
```

}

Output:



Lab3:

Program:

#include <stdio.h>

int main() {

int n, i, j;

int bt[10], pr[10], p[10];

int wt[10], tat[10], temp;

printf("Enter number of processes: ");

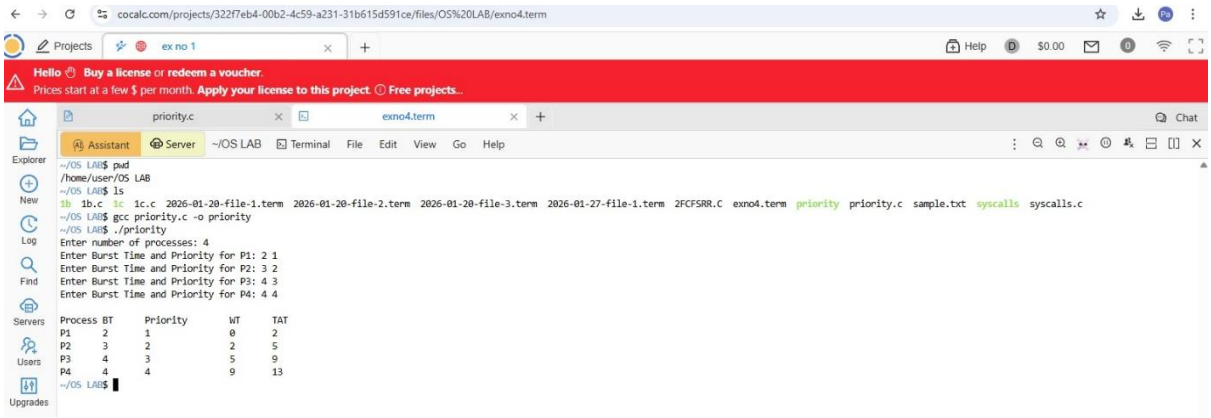
scanf("%d", &n);

```
for(i = 0; i < n; i++) {
p[i] = i + 1;
printf\




  Enter Burst Time and Priority for P%d: ", p[i]);
scanf("%d %d", &bt[i], &pr[i]);
}
// Sort by Priority
for(i = 0; i < n; i++) {
for(j = i + 1; j < n; j++) {
if(pr[i] > pr[j]) {
temp = pr[i]; pr[i] = pr[j]; pr[j] = temp;
temp = bt[i]; bt[i] = bt[j]; bt[j] = temp;
temp = p[i]; p[i] = p[j]; p[j] = 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];
printf("\nProcess\tBT\tPriority\tWT\tTAT\n");
for(i = 0; i < n; i++)
printf("P%d\t%d\t%d\t\t%d\t%d\n", p[i], bt[i], pr[i], wt[i], tat[i]);
```

return 0;

}

Output:



Lab4:

Program:

#include <stdio.h>

#include <pthread.h>

#include <semaphore.h>

#include <unistd.h>

#define N 5

sem_t forks[N];

pthread_t philosophers[N];

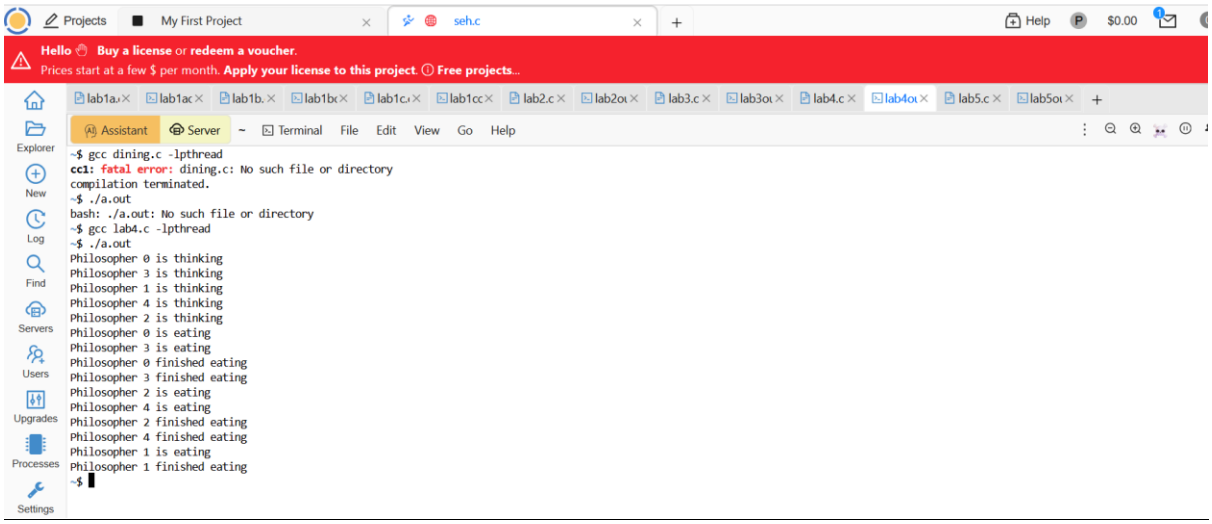
void* philosopher(void* num) {

int id = *(int*)num;

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

```
sleep(1);
sem_wait(&forks[id]);
sem_wait(&forks[(id + 1) % N]);
  printf("Philosopher %d is eating\n", id);
  sleep(1);
  sem_post(&forks[id]);
  sem_post(&forks[(id + 1) % N]);
  printf("Philosopher %d finished eating\n", id);
  return NULL;
}
int main() {
  int i, id[N];

  for (i = 0; i < N; i++)
    sem_init(&forks[i], 0, 1);
  for (i = 0; i < N; i++) {
    id[i] = i;
    pthread_create(&philosophers[i], NULL, philosopher, &id[i]);
  }
  for (i = 0; i < N; i++)
    pthread_join(philosophers[i], NULL);
  return 0;
}
```

Output:



Lab5:

Program:

```
#include<stdio.h>

void firstFit(int blockSize[], int m, int processSize[], int n)

{

int alloca on[n];

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

    alloca on[i]=-1;


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

  {

    for(int j=0;j<m;j++)

    {

      if(blockSize[j] >= processSize[i])

      {

        alloca on[i] = j;

        blockSize[j] -= processSize[i];

        break;
```

```
        }
    }
  }

  prin ("\nFIRST FIT\nProcess No.\tProcess Size\tBlock No.\n");
  for(int i=0;i<n;i++)
  {
    if(alloca on[i]!=-1)
      prin ("%d\t\t%d\t\t%d\n",i+1,processSize[i],alloca on[i]+1);
    else
      prin ("%d\t\t%d\t\tNot Allocated\n",i+1,processSize[i]);
  }
}

void bestFit(int blockSize[], int m, int processSize[], int n)
{
  int alloca on[n];
  for(int i=0;i<n;i++)
    alloca on[i]=-1;

  for(int i=0;i<n;i++)
  {
    int bestIdx=-1;
    for(int j=0;j<m;j++)
    {
      if(blockSize[j]>=processSize[i])
      {
        if(bestIdx==-1 || blockSize[j]<blockSize[bestIdx])
```

```
            bestIdx=j;
        }
    }

    if(bestIdx!=-1)
    {
        alloca on[i]=bestIdx;
        blockSize[bestIdx]-=processSize[i];
    }
  }

  prin ("\nBEST FIT\nProcess No.\tProcess Size\tBlock No.\n");
  for(int i=0;i<n;i++)
  {
    if(alloca on[i]!=-1)
        prin ("%d\t\t%d\t\t%d\n",i+1,processSize[i],alloca on[i]+1);
    else
        prin ("%d\t\t%d\t\tNot Allocated\n",i+1,processSize[i]);
  }
}

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

  int m = 5;
  int n = 4;
```

```
    int block1[5], block2[5];
    for(int i=0;i<m;i++)
    {
        block1[i]=blockSize[i];
        block2[i]=blockSize[i];
    }

    firstFit(block1,m,processSize,n);
    bestFit(block2,m,processSize,n);

    return 0;
}
```

Output: