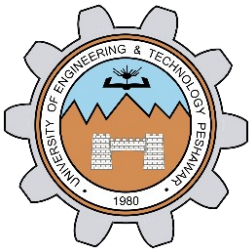
PROCESS SYNCHRONIZATION

AND SCHEDULING

LAB # 8



Spring 2020

CSE204L Operating Systems Lab

Submitted by: **SHAH RAZA**

Registration No. : **18PWCSE1658**

Class Section: **B**

"On my honor, as student of University of Engineering and Technology, I have neither given nor received unauthorized assistance on this academic work."

Student Signature: _____

Submitted to:

Engr. Mian Ibad Ali Shah

Tuesday, 21 July, 2020

Department of Computer Systems Engineering

University of Engineering and Technology, Peshawar

Lab Objective(s):

- Solve the Critical Section Problem
- Implement different Process Scheduling Algorithms.

Task # 01:

Solve the critical section problem using:

i. Peterson solution
ii. Test and set instruction
iii. producer/consumer problem

Peterson Solution:

Code:

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

void *func1(void *);
void *func2(void *);

int flag[2];
int turn=0;
int global=100;


int main()
{
    pthread_t tid1,tid2;
    pthread_create(&tid1,NULL,func1,NULL);
    pthread_create(&tid2,NULL,func2,NULL);
    pthread_join(tid1,NULL);
    pthread_join(tid2,NULL);
}

void *func1(void *param)
{
    int i=0;
    while(i<2)
    {
        flag[0]=1;
        turn=1;
        while(flag[1]==1 && turn==1);
        global+=100;
        flag[0]=0;
        i++;
        printf("First thread: Global: %d\n",global);
    }
}
```

```c
void *func2(void *param)
{
    int i=0;
    while(i<2)
    {
        flag[1]=1;
        turn=0;
        while(flag[0]==1 && turn==0);
        global-=25;
        flag[1]=0;
        i++;
        printf("Second thread: Global: %d\n",global);
    }
}
```

Output :

```
ShahRaza@ubuntu:~/Work/OS/Lab 8/Task 1/Peterson Solution$ gcc Peterson.c -o Peterson -lpthread
ShahRaza@ubuntu:~/Work/OS/Lab 8/Task 1/Peterson Solution$ ./Peterson
Second thread: Global: 75
Second thread: Global: 50
First thread: Global: 150
First thread: Global: 250
```

Test And Set Instruction:

Code:

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

void *func1(void *);
void *func2(void *);

int Lock=0;
int global=100;

int main()
{
    pthread_t tid1,tid2;
    pthread_create(&tid1,NULL,func1,NULL);
    pthread_create(&tid2,NULL,func2,NULL);
    pthread_join(tid1,NULL);
    pthread_join(tid2,NULL);
}

int TestAndSet(int *Lock)
{
    int temp= *Lock;
    *Lock=1;
    return temp;
}
```

```c
void *func1(void *param)
{
    int i=0;
    while(i<2)
    {
        while(TestAndSet(&Lock));
        global+=100;
        Lock=0;
        i++;
        printf("First Thread: Global: %d\n",global);
    }

}

void *func2(void *param)
{
    int i=0;
    while(i<2)
    {
        while(TestAndSet(&Lock));
        global-=25;
        Lock=0;
        i++;
        printf("Second Thread: Global: %d\n",global);
    }
}
```

Output :

```
ShahRaza@ubuntu:~/Work/OS/Lab 8/Task 1/Test and Set Instruction$ gcc TSL.c -o TSL -lpthread
ShahRaza@ubuntu:~/Work/OS/Lab 8/Task 1/Test and Set Instruction$ ./TSL
Second Thread: Global: 75
Second Thread: Global: 50
First Thread: Global: 150
First Thread: Global: 250
```

Producer/Consumer Problem:

Code:

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

void *Producer(void *);
void *Consumer(void *);

void wait(int *s)
{
    while(s<=0);
    s--;
}

void signal(int *s)
{
    s++;
}


int empty=1,full=0,sm=1;
int global=100;

int main()
{
    pthread_t ptid,ctid;
    pthread_create(&ptid,NULL,Producer,NULL);
    pthread_create(&ctid,NULL,Consumer,NULL);
    pthread_join(ptid,NULL);
    pthread_join(ctid,NULL);
}
```

```c
void *Producer(void *param)
{
    int i=0;
    while(i<2)
    {
        wait(&empty);
            wait(&sm);
                global+=100;
            signal(&sm);
        signal(&full);
        i++;
        printf("Producer: Global: %d\n",global);
    }

}

void *Consumer(void *param)
{
    int i=0;
    while(i<2)
    {
        wait(&full);
            wait(&sm);
                global-=25;
            signal(&sm);
        signal(&empty);
        i++;
        printf("Consumer: Global: %d\n",global);
    }
}
```

Output :

```
ShahRaza@ubuntu:~/Work/OS/Lab 8/Task 1/Producer Consumer Problem$ gcc PCP.c -o PCP -lpthread
ShahRaza@ubuntu:~/Work/OS/Lab 8/Task 1/Producer Consumer Problem$ ./PCP
Consumer: Global: 75
Consumer: Global: 50
Producer: Global: 150
Producer: Global: 250
```

Task # 02:

Simulate the following Process Scheduling Algorithms:

i. First Come First Serve
ii. Shortest Job First
iii. Priority Scheduling
iv. Shortest Time Remaining First

First Come First Serve:

Code:

```c
#include <stdio.h>
const int pid=0;
const int arrival=1;
const int burst=2;
const int start=3;
const int end=4;
const int wait=5;
const int tat=6;
const int info=7;
const int num_p=4;
int main()
{
    int process[num_p][info],i;
    for(i=0;i<num_p;i++)
    {
        process[i][pid]=i;
        process[i][arrival]=i;
        printf("Enter the Burst time for Process %d: \n",i);
        scanf("%d",&process[i][burst]);
        if(!i)
            process[i][start]=0;
        else
            process[i][start]=process[i-1][end];
        process[i][end]=process[i][start]+process[i][burst];
        process[i][wait]=process[i][start]-process[i][arrival];
        process[i][tat]=process[i][wait]+process[i][burst];
    }
    printf("PID Arrival Burst   Start   End Wait    TAT\n");
    for(i=0;i<num_p;i++)
    {
        printf("%d \t %d \t %d \t %d \t %d \t %d \t %d \n",process[i][pid],process[i][arrival],
        process[i][burst],process[i][start],process[i][end],process[i][wait],process[i][tat]);
    }
}
```

Output :

```
ShahRaza@ubuntu:~/Work/OS/Lab 8/Task 2/FCFS$ gcc FCFS.c -o FCFS
ShahRaza@ubuntu:~/Work/OS/Lab 8/Task 2/FCFS$ ./FCFS
Enter the Burst time for Process 0:
10
Enter the Burst time for Process 1:
5
Enter the Burst time for Process 2:
15
Enter the Burst time for Process 3:
3
PID     Arrival Burst   Start   End     Wait    TAT
0        0       10      0       10      0       10
1        1       5       10      15      9       14
2        2       15      15      30      13      28
3        3       3       30      33      27      30
```

Shortest Job First:

Code:

```c
#include <stdio.h>
const int pid=0;
const int arrival=1;
const int burst=2;
const int start=3;
const int end=4;
const int wait=5;
const int tat=6;
const int info=7;
const int num_p=4;
int main()
{
    int process[num_p][info],i,j;
    for(i=0;i<num_p;i++)
    {
        process[i][pid]=i;
        process[i][arrival]=i;
        printf("Enter the Burst time for Process %d: \n",i);
        scanf("%d",&process[i][burst]);
        for(j=i;j>=0;j--)
        {
            if(process[j][pid]==0)
                continue;
            else
            {
                if(process[j][burst]<process[j-1][burst])
                {
                    int temp=process[j-1][burst];
                    process[j-1][burst]=process[j][burst];
                    process[j][burst]=temp;
                }
            }
        }
    }
```

```
    for(i=0;i<num_p;i++)
    {
        if(!i)
            process[i][start]=0;
        else
            process[i][start]=process[i-1][end];
        process[i][end]=process[i][start]+process[i][burst];
        process[i][wait]=process[i][start]-process[i][arrival];
        process[i][tat]=process[i][wait]+process[i][burst];
    }

    printf("PID Arrival Burst   Start   End Wait    TAT\n");

    for(i=0;i<num_p;i++)
    {
        printf("%d \t %d \t %d \t %d \t %d \t %d \t %d \n",process[i][pid],process[i][arrival],
        process[i][burst],process[i][start],process[i][end],process[i][wait],process[i][tat]);
    }
}
```

Output:

```
ShahRaza@ubuntu:~/Work/OS/Lab 8/Task 2/SJF$ gcc SJF.c -o SJF
ShahRaza@ubuntu:~/Work/OS/Lab 8/Task 2/SJF$ ./SJF
Enter the Burst time for Process 0:
4
Enter the Burst time for Process 1:
7
Enter the Burst time for Process 2:
3
Enter the Burst time for Process 3:
9
PID     Arrival Burst   Start   End     Wait    TAT
0        0       3       0       3       0       3
1        1       4       3       7       2       6
2        2       7       7       14      5       12
3        3       9       14      23      11      20
```

Priority Scheduling:

Code:

```c
#include <stdio.h>
const int pid=0;
const int priority=1;
const int arrival=2;
const int burst=3;
const int start=4;
const int end=5;
const int wait=6;
const int tat=7;
const int info=8;
const int num_p=4;
int main()
{
    int process[num_p][info],i,j;
    for(i=0;i<num_p;i++)
    {
        process[i][pid]=i;
        process[i][arrival]=i;
        printf("Enter the Burst time for Process %d: \n",i);
        scanf("%d",&process[i][burst]);
        printf("Enter the Priority for Process %d: \n",i);
        scanf("%d",&process[i][priority]);
        for(j=i;j>=0;j--)
        {
            if(process[j][pid]==0)
                continue;
            else if(process[j][priority]<process[j-1][priority])
            {
                int temp1=process[j-1][priority];
                process[j-1][priority]=process[j][priority];
                process[j][priority]=temp1;
                int temp2=process[j-1][burst];
                process[j-1][burst]=process[j][burst];
                process[j][burst]=temp2;
                int temp3=process[j-1][pid];
                process[j-1][pid]=process[j][pid];
                process[j][pid]=temp3;
            }
        }
    }
    for(i=0;i<num_p;i++)
    {
        if(!i)
            process[i][start]=0;
        else
            process[i][start]=process[i-1][end];
        process[i][end]=process[i][start]+process[i][burst];
        process[i][wait]=process[i][start]-process[i][arrival];
        process[i][tat]=process[i][wait]+process[i][burst];
    }
    printf("PID Priority    Arrival Burst   Start   End Wait    TAT\n");
    for(i=0;i<num_p;i++)
    {
        printf("%d \t %d \t %d \t %d \t %d \t %d \t %d \t %d \n",process[i][pid],process[i][priority],
        process[i][arrival],process[i][burst],process[i][start],process[i][end],process[i][wait],process[i][tat]);
    }
}
```

Output:

```
ShahRaza@ubuntu:~/Work/OS/Lab 8/Task 2/Priority Scheduling$ gcc Priority.c -o Priority
ShahRaza@ubuntu:~/Work/OS/Lab 8/Task 2/Priority Scheduling$ ./Priority
Enter the Burst time for Process 0:
5
Enter the Priority for Process 0:
2
Enter the Burst time for Process 1:
3
Enter the Priority for Process 1:
4
Enter the Burst time for Process 2:
10
Enter the Priority for Process 2:
1
Enter the Burst time for Process 3:
8
Enter the Priority for Process 3:
3
PID     Priority Arrival Burst Start     End      Wait     TAT
2        1       0       10      0       10       0        10
0        2       1       5       10      15       9        14
3        3       2       8       15      23       13       21
1        4       3       3       23      26       20       23
```

Shortest Remaining Time First:

Code:

```c
#include <stdio.h>
const int pid=0;
const int arrival=1;
const int burst=2;
const int start=3;
const int end=4;
const int remain_time=5;
const int wait=6;
const int tat=7;
const int info=8;
const int num_p=4;
int main()
{
    int process[num_p][info],i,j,time,complete=0;
    int minm=9999,smallest=0,check=0;
    for(i=0;i<num_p;i++)
    {
        process[i][pid]=i;
        printf("Enter the Burst time for Process %d: \n",i);
        scanf("%d",&process[i][burst]);
        printf("Enter the Arrival time for Process %d: \n",i);
        scanf("%d",&process[i][arrival]);
        process[i][remain_time]=process[i][burst];
    }
    for(time=0;complete!=num_p;time++)
    {
        for(j=0;j<num_p;j++)
        {
            if(process[j][arrival]<=time && process[j][remain_time]<minm && process[j][remain_time]>0)
            {
                minm= process[j][remain_time];
                smallest=j;
                check=1;
            }
        }
```

```
        if(check==0)
            continue;
        process[smallest][remain_time]--;
        minm=process[smallest][remain_time];
        if(minm==0)
            minm=9999;
        if(process[smallest][remain_time]==0)
        {
            complete++;
            check=0;
            process[smallest][end]=time+1;
            process[smallest][wait]=process[smallest][end]-process[smallest][burst]-process[smallest][arrival];
            if(process[smallest][wait]<0)
                process[smallest][wait]=0;
        }
    }
    for(i=0;i<num_p;i++)
        process[i][tat]=process[i][wait]+process[i][burst];
    printf("PID Arrival Burst   End Wait    TAT\n");
    for(i=0;i<num_p;i++)
    {
        printf("%d \t %d \t %d \t %d \t %d \t %d \n",process[i][pid],process[i][arrival],
        process[i][burst],process[i][end],process[i][wait],process[i][tat]);
    }
}
```

Output:

```
ShahRaza@ubuntu:~/Work/OS/Lab 8/Task 2/SRTF$ gcc SRTF.c -o SRTF
ShahRaza@ubuntu:~/Work/OS/Lab 8/Task 2/SRTF$ ./SRTF
Enter the Burst time for Process 0:
10
Enter the Arrival time for Process 0:
3
Enter the Burst time for Process 1:
2
Enter the Arrival time for Process 1:
4
Enter the Burst time for Process 2:
5
Enter the Arrival time for Process 2:
2
Enter the Burst time for Process 3:
6
Enter the Arrival time for Process 3:
7
PID     Arrival Burst   End     Wait    TAT
0        3       10      25      12      22
1        4       2       6       0       2
2        2       5       9       2       7
3        7       6       15      2       8
```