AU1940211 Samyak Kothari

OS Lab Exam

Q1 – B

```
//Samyak Kothari AU1940211
#include<pthread.h>
#include<stdio.h>
#include<stdlib.h>
#include<semaphore.h>
sem_t semaphore;
int l = 0;
void *thread_function(void *arg){
   int i;
   for(i=0;i<26;i++){
      sem_wait(&semaphore);
      if(l%3 != 1){
         i--;
      }else{
         printf("%d ",1+i);
         l++;
      }
      sem_post(&semaphore);
   }
   pthread_exit(NULL);
}
void *thread_function2(void *arg){
   int i;
   for(i=0;i<26;i++){
      sem_wait(&semaphore);
      if(l%3 != 0){
         i--;
```

```
        }else{
            printf("%c ",'A'+i);
            l++;
        }
        sem_post(&semaphore);
    }
    pthread_exit(NULL);
}
void *thread_function3(void *arg){
    int i;
    for(i=0;i<26;i++){
        sem_wait(&semaphore);
        if(l%3 != 2){
            i--;
        }else{
            printf("%c ",'a'+i);
            l++;
        }
        sem_post(&semaphore);
    }
    pthread_exit(NULL);
}

int main(){
    pthread_t thread1, thread2, thread3;
    char start1 = 'A';
    char start2 = 'a';
    sem_init(&semaphore, 0, 1);
    pthread_create(&thread1,NULL,thread_function2,NULL);
    pthread_create(&thread2,NULL,thread_function,NULL);
    pthread_create(&thread3,NULL,thread_function3,NULL);
```

```
    pthread_join(thread1,NULL);
    pthread_join(thread2,NULL);
    pthread_join(thread3,NULL);
    sem_destroy(&semaphore);
    return 0;
}
```

Q2 - A

Round robin –

Code –

```
//Samyak Kothari AU1940211
#include<stdio.h>
int main()
{
    int i, limit, total = 0, x, counter = 0, time_q;
    int wait_t = 0, tat = 0, arrival_time[10], burst_t[10], temp[10];
    float avg_wait, avg_tat;
    printf("\nEnter Number of Processes:");
    scanf("%d", &limit);
    x = limit;
    for(i = 0; i < limit; i++)
    {
        printf("\nEnter all Details of Process[%d]\n", i + 1);
        printf("Arrival Time -");
        scanf("%d", &arrival_time[i]);
        printf("Burst Time - ");
        scanf("%d", &burst_t[i]);
        temp[i] = burst_t[i];
    }
```

```
    printf("\nEnter Time Quantum - ");
    scanf("%d", &time_q);
    printf("\nProcess ID\t\tBurst Time\t Turnaround Time\tWaiting Time\n");
    for(total = 0, i = 0; x != 0;)
    {
        if(temp[i] <= time_q && temp[i] > 0)
        {
            total = total + temp[i];
            temp[i] = 0;
            counter = 1;
        }
        else if(temp[i] > 0)
        {
            temp[i] = temp[i] - time_q;
            total = total + time_q;
        }
        if(temp[i] == 0 && counter == 1)
        {
            x--;
            printf("\nP[%d]\t\t%d\t\t %d\t\t\t %d", i + 1, burst_t[i], total - arrival_time[i], total -
arrival_time[i] - burst_t[i]);
            wait_t = wait_t + total - arrival_time[i] - burst_t[i];
            tat = tat + total - arrival_time[i];
            counter = 0;
        }
        if(i == limit - 1)
        {
            i = 0;
        }
        else if(arrival_time[i + 1] <= total)
        {
```

```
            i++;
        }
        else
        {
            i = 0;
        }
    }

    avg_wait = wait_t * 1.0 / limit;
    avg_tat = tat * 1.0 / limit;
    printf("\nAverage Waiting Time-\t%f", avg_wait);
    printf("\nAvg Turnaround Time-\t%fn", avg_tat);
    return 0;
}
```

Modified round robin –

```
//Samyak Kothari AU1940211
#include <stdio.h>
 #include <conio.h>
 #include <stdio.h>
 #include <conio.h>
 #include<math.h>
 #include<string.h>
 int wait_t[100],burst_t[100],at[100],tat[100],n,p[100];
 float avg_wait[5],avg_tat[5];
 int temp1,temp2,temp3,sqt,avg;
int main(){
            printf("Enter Number of processes - ");
       scanf("%d",&n);
       int x;
                    for(x=0;x<n;x++)
                                p[x]=x+1;
```

```
for(x=0;x<n;x++)
{
  printf("Enter Burst Time of Process [%d] - ",x+1);
  scanf("%d",&burst_t[x]);
  printf("Enter Arrival Time of process [%d] - ",x+1);
  scanf("%d",&at[x]);
}
                for(x=0;x<5;x++)
                {
                        avg_wait[x]=0.0;
                        avg_tat[x]=0.0;
                }
     int burst_t1[n],i,j,temp,qt;
     int b[n];
     float twait_t,ttat;
     for(i=0;i<n;i++)
                burst_t1[i]=burst_t[i];
     for(i=0;i<n;i++)
                b[i]=burst_t[i];
     int num=n;
     int time=0;
     int max;
     int sum,t,a,ap;
     ap=0;
for (i = 0; i < n; i++)
{
   for (j = 0; j < n-i- 1; j++)
   {
      if (burst_t[j] > burst_t[j + 1])
      {
          temp1 = burst_t[j];
```

```
            temp2 = p[j];
            temp3 = at[j];
            burst_t[j] = burst_t[j + 1];
            p[j] = p[j + 1];
            at[j] = at[j + 1];
            burst_t[j + 1] = temp1;
            p[j + 1] = temp2;
            at[j + 1] = temp3;
        }
    }
 }
max=burst_t[n-1];
sum=0;
for(i=0;i<n;i++)
{
   sum=sum+burst_t[i];
}
avg=sum/n;

qt=(avg+max)/2;
printf("\nCalculated Dynamic Quantum time  - %d\n",qt);

                while(num>0){
                a=0;
                max=0;
                sum=0;
                t=0;
                for(i=0;i<n;i++){
                        if(at[i]<=time && b[i]!=0)
                        {
                                if(b[i]<qt)
```

```
                                {
                                        t+=b[i];
                                        b[i]=0;
                                }
                                else
                                {
                                        t+=qt;
                                        b[i]-=qt;
                                }
                                if(b[i]<qt && b[i]!=0)
                                {
                                        t+=b[i];
                                        b[i]=0;
                                }
                                if(b[i]==0){
                                        wait_t[i]=(time+t)-burst_t1[i];
                                        tat[i]=time+t;
                                        num--;
                                }
                        }
                }
                time+=t;
                }
                printf("Processes ID\tWaiting time\tTurnAround Time\n");
                for(j=1;j<=n;j++)
{
   for(i=0;i<n;i++)
   {
      if(j==p[i])
                                        printf("P[%d]\t\t%d\t\t\t%d\n",p[i],wait_t[i],tat[i]);
                                }
```

```
                        }

    twait_t=0;
    ttat=0;
                for(i=0;i<n;i++)
                        {twait_t=twait_t+wait_t[i];}
                avg_wait[4]=twait_t/n;
                for(i=0;i<n;i++)
                        {ttat=ttat+tat[i];}
                avg_tat[4]=(ttat/n);
}
```

Q3

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

sem_t empty;
sem_t full;
int in = 0;
int i;
int max_I = 5;
int BufferSize = 5;
int out = 0;
int buffer[BufferSize];
pthread_mutex_t mutex;

void *producer(void *p)
{
   int item;
```

```
    for(i = 0; i < max_I; i++) {
        item = rand();
        sem_wait(&empty);
        pthread_mutex_lock(&mutex);
        buffer[in] = item;
        printf("Producer %d: Insert Item %d at %d\n", *((int *)p),buffer[in],in);
        in = (in+1)%BufferSize;
        pthread_mutex_unlock(&mutex);
        sem_post(&full);
    }
}
void *consumer(void *c)
{
    for(i = 0; i < max_I; i++) {
        sem_wait(&full);
        pthread_mutex_lock(&mutex);
        int item = buffer[out];
        printf("Consumer %d: Remove Item %d from %d\n",*((int *)c),item, out);
        out = (out+1)%BufferSize;
        pthread_mutex_unlock(&mutex);
        sem_post(&empty);
    }
}

int main()
{

    pthread_t pro[5],con[5];
    pthread_mutex_init(&mutex, NULL);
    sem_init(&empty,0,BufferSize);
    sem_init(&full,0,0);
```

```
    int a[5] = {1,2,3,4,5};
    for(i = 0; i < 5; i++) {
        pthread_create(&pro[i], NULL, (void *)producer, (void *)&a[i]);
    }
    for(i = 0; i < 5; i++) {
        pthread_create(&con[i], NULL, (void *)consumer, (void *)&a[i]);
    }

    for(i = 0; i < 5; i++) {
        pthread_join(pro[i], NULL);
    }
    for(i = 0; i < 5; i++) {
        pthread_join(con[i], NULL);
    }

    pthread_mutex_destroy(&mutex);
    sem_destroy(&empty);
    sem_destroy(&full);

    return 0;

}
```