Yasemin Akın
2210782

CS342 OPERATING SYSTEMS
HOMEWORK 2

Q1. 510 700
510 780
620 150
1620 150

Q2. 1300 3000 5300
1000 3000 400

Q3. 51 21 2
51 23 3
51 24 4
51 22 1
51 25 5
10 25 5

* Mutex lock used here can cause inconsistent outputs between compilations, the reason why the order of outputs may change.

Q4. RR (q=10)

	A	B	C	D	E
Finish (completion) time	355	345	385	325	305
Turnaround time	355	330	360	280	240
Waiting time	225	250	260	220	195
Response time	0	5	15	25	35

→ 313 on average

SJF

	A	B	C	D	E
Finish time	100	285	385	205	145
Turnaround time	100	270	360	160	80
Waiting time	0	190	260	100	35
Response time	0	190	260	100	35

→ 194 on average

FCFS

	A	B	C	D	E
Finish T.	100	180	280	340	385
Turnaround	100	165	255	295	320
Waiting	0	85	155	235	275
Response	0	85	155	235	275

→ 227 on average

Yasemin Akın 22101782

SRTF

	A	B	C	D	E
Finish Time	285	95	385	200	140
TurnaraindT	285	80	360	155	75
waiting T	185	0	260	95	3
ResponseT.	0	0	260	95	30

→ 191 on average

Comparison of Average Turnaround Times:
RR > FCFS > SJF > SRTF

Q6. a) CPU utilization is the fraction of time the CPU is busy.
$P = \lambda/\mu = 25/28 \cong 0.893 \rightarrow 89.3\%$ (40ms = 25/s.)

b) P_0, is the complement of CPU utilization.
$P_0 = 1 - P \cong 1 - 0.893 = 0.107 \rightarrow 10.7\%$

c) $E[R]$ is the inverse of the difference between μ and λ
$E[R] = 1/(\mu - \lambda) = 1/28 - 25 = 1/3\ s \cong 333.33$ ms

d) $L_q = P^2/(1-P) \cong 0.893^2/(1-893) \cong 7.45$

e) $W_q = L_q/\lambda \cong 7.45/25 = 0.298\ s = 298$ ms

f) "$E[R]' = 166.67$ ms" is what is wanted
$E[R]' = 1/(\mu - \lambda')$, $\lambda' = 22$

Q7. Disadvantages:

① CRASH RISK IS HIGH : Since all user-space threads use the same memory areas, if one of them crashes, it will destroy the others' work too kernel-space threads on the other hand are more isolated, making them more secure.

② Input/Output Blocking : 1 thread in user-space doing I/O operation is able to freeze the whole program since OS sees our program as 1 complete chunk. That is not an issue on kernel-space since OS manages each thread in seperate manner.

Advantage:

① Speed: more high because system does not spend time to talk to kernel.

② More adaptable: Threads in user-space can easily be customized according to users' programs needs, rules can be changed to decide how threads behave and interact.

Yasemin Akın 22101782

Q8.

```
void request() {
    pthread_mutex_lock(&mutex);
    while (current_count >= N) {
        pthread_cond_wait(&cond_var, &mutex);
    }
    current_count++;
    pthread_mutex_unlock(&mutex);
}
void release(){
    pthread_mutex_lock(&mutex);
    current_count--;
    pthread_cond_signal(&cond_var);
    pthread_mutex_unlock(&mutex);
}
```

&mutex → our mutex that has been created

N → maximum allowed number of threads

&cond_var → condition variable

current_count → represent current count of threads

Q6.

TIME	PROCESSNAME	ACTION	QUEUE	ACTION
0	A	E	H	
1	B	E	H	
2	C	E	H	moves to M
3	A	E	M	goes to I/O
5	B	E	M	goes to I/O
7	C	E	M	continue in M
9	C	E	M	
11	C	E	M	moves to L
15	C	E	L	
19	C	E	L	
23	C	E	L	
25	A B C	PB	H	all back to H
26	A	R I/O	H	
27	B	R I/O	H	
28	A	E	H	
29	B	E	H	
30	C	E	H	

E: execute

PB: priority boost

R I/O: return from I/O operation

Yasemin Akın 2210782

Q9. class GeneralSemaphore :

```
int val;
BinarySemaphore mutex;
BinarySemaphore condVar;

generalSemaphoreInitalize (GS, initalVal)
    GS.val = InitalVal;
    binarySemaphoreInitalize (GS.mutex, 1);
    binarySemaphoreInitalize (GS.condVar, 0);

generalSemaphoreRequest (GS)
    binarySemaphoreRequest (GS.mutex);
    GS.val--;
    if GS.val < 0 then
        binarySemaphoreRelease (GS.mutex)
        binarySemaphoreRequest (GS.condVar)
    else
        binarySemaphoreRelease (GS.mutex)

generalSemaphoreRelease (GS)
    binarySemaphoreRequest (GS.mutex);
    GS.val++
    if (GS.val <= 0) then
        binarySemaphoreRelease (GS.condVar)
    binarySemaphoreRelease (GS.mutex)
```

NOTE:

— binarySemaphoreRelease
— binarySemaphoreInitialize
— binarySemaphoreRequest

} methods are accepted to be a part of BinarySemaphore class provided by the OS.