COMP 2432 Operating Systems

Solution to Written Assignment 1

1. CPU Scheduling.

(a) SRT

```
12233333222222111111111444444444445555555555555
```

Pid	Burst	Arr	Prior	Wait	TR
1	11	0	4	13	24
2	8	1	2	5	13
3	5	3	1	0	5
4	12	6	5	18	30
5	14	8	3	28	42

(b) Priority with Preemption (Linux)

```
12233333222222555555555555551111111111444444444444
```

Pid	Burst	Arr	Prior	Wait	TR
1	11	0	4	27	38
2	8	1	2	5	13
3	5	3	1	0	5
4	12	6	5	32	44
5	14	8	3	6	20

(c) Priority with Preemption (Windows)

```
11111144444444444411111555555555555552222222233333
```

Pid	Burst	Arr	Prior	Wait	TR
1	11	0	4	12	23
2	8	1	2	36	44
3	5	3	1	42	47
4	12	6	5	0	12
5	14	8	3	15	29

(d) RR, q = 5

```
11111222223333311111444445555522214444455555445555
```

Pid	Burst	Arr	Prior	Wait	TR
1	11	0	4	23	34
2	8	1	2	24	32
3	5	3	1	7	12
4	12	6	5	28	40
5	14	8	3	28	42

(e) RR, q = 3

```
11122211133322244455511133224445551144455544455555
```

Pid	Burst	Arr	Prior	Wait	TR
1	11	0	4	25	36
2	8	1	2	19	27
3	5	3	1	18	23
4	12	6	5	27	39
5	14	8	3	28	42

(f) SRT, T = 1 (t refers to context switching overhead)

```
1t2t33t3t33t2222222t1111111111t444444444444t55555555555555
```

SRT, T = 2

```
1tt333tt33tt22222222tt1111111111tt444444444444tt55555555555555
```

Another possible answer: since context switching decision was already made at time 1 when P2 arrives, the overhead between time 1 to 3 would be for admitting P2. P3 only arrives at time 3 and P2 is already ready for execution. If P3 is not switched in here, it can only try to start at time 4 and actually time 6, though it has a shorter remaining time, leading to another context switching later.

```
1tt2tt33tt333tt2222222tt1111111111tt444444444444tt55555555555555
```

Turnaround time: 34 21 10 42 56, average 32.6.

PR-Linux, T = 1

```
1t2t33t3t33t2222222t55555555555555t1111111111t444444444444
```

PR-Linux, T = 2

```
1tt333tt33tt22222222tt55555555555555tt1111111111tt444444444444
```

Another possible answer: since context switching decision was already made at time 1 when P2 arrives, the overhead between time 1 to 3 would be for admitting P2. P3 only arrives at time 3 and P2 is already ready for execution. If P3 is not switched in here, it can only try to start at time 4 and actually time 6, though it has a higher priority, leading to another context switching later.

```
1tt2tt33tt333tt2222222tt55555555555555tt1111111111tt444444444444
```

Turnaround time: 50 21 10 58 30, average 33.8.

PR-Win, T = 1

```
1t1t11t4t44444444444t1111111t55555555555555t22222222t33333
```

PR-Win, T = 2

```
1tt111tt444444444444tt1111111tt55555555555555tt22222222tt33333
```

We do not have a second possible answer for this case, since P4 arrives at time 6 and the decision is to switch to P4 at the end of context switching at time 8.

RR, q = 5, T = 1

```
11111t22222t33333t11111t44444t55555t222t1t44444t55555t44t5555
```

RR, q = 5, T = 2

```
11111tt22222tt33333tt11111tt44444tt55555tt222tt1tt44444tt55555tt44tt5555
```

RR, q = 3, T = 1

```
111t222t111t333t444t222t555t111t33t444t22t555t11t444t555t444t555t55
```

RR, q = 3, T = 2

```
111tt222tt111tt333tt444tt222tt555tt111tt33tt444tt22tt555tt11tt444tt555tt444tt555tt55
```

(*f*)	SRT			PR-Linux			PR-Win			RR-5			RR-3		
	T=0	*T*=1	*T*=2	*T*=0	*T*=1	*T*=2	*T*=0	*T*=1	*T*=2	*T*=0	*T*=1	*T*=2	*T*=0	*T*=1	*T*=2
P_1	24	30	32	38	45	48	23	28	29	34	41	48	36	48	60
P_2	13	18	19	13	18	19	44	51	54	32	38	44	27	40	50
P_3	5	8	7	5	8	7	47	55	59	12	14	16	23	31	39
P_4	30	37	40	44	52	56	12	14	14	40	50	60	39	54	69
P_5	42	50	54	20	26	28	29	35	37	42	53	64	42	59	76
Average	22.8	28.6	30.4	24.0	29.8	31.6	31.0	36.6	38.6	32.0	39.2	46.4	33.4	46.4	58.8

Some possible observations: It is natural that with increasing value of *T*, the average turnaround time for all algorithms increases, since additional time costs will incur in the form of context switching. However, if the individual processes are concerned, the increasing trend is still valid for majority of cases. There could be few exceptions, e.g. P_3 in **SRT**, **PR-Linux**, namely, being the first short/high-priority process that may *jump over* the queue to complete earlier, or not be affected, e.g. high priority process P_4 in **PR-Win**. It can be seen that the processes are scheduled in the roughly the same order with roughly the same number of context switchings in these algorithms, with more potential deviations in **SRT** and **PR-Linux**.

2. Multi-level Scheduling.

Fixed Priority Scheduling.

```
Multi-level Feedback Queue: fixed priority
Processes on queue 1 can begin (a space per 5 time units for easier viewing)
11221 13322 44335 54455
Queue 1 becomes empty at time 20
Processes on queue 2 can now begin
                        11112 22234 44455 55111 44445 555
Queue 2 becomes empty at time 48
Processes on queue 3 can now begin
                                                          55
Queue 3 becomes empty at time 50
```

```
Summarizing
Q1: 11221 13322 44335 54455
Q2:                          11112 22234 44455 55111 44445 555
Q3:                                                              55
```

Pid	Burst	Arr	Prior	Wait	TR
1	11	0	4	29	40
2	8	1	2	19	27
3	5	3	1	21	26
4	12	6	5	26	38
5	14	8	3	28	42
Avg.				24.6	34.6

Time Slicing Scheduling.

```
Multi-level Feedback Queue: time slicing
Pay attention to the moment when each queue uses up its allocated time
Processes on queue 1 can begin (a space per 5 time units for easier viewing)
11221 1
Queue 1 uses up its time slice at time 6
Processes on queue 2 can now begin, only P1 is there
       1111 111
Queue 2 completes all its processes at time 13
Queue 3 is empty at time 13

Queue 1 continues to use its next time slice
               33 2244
Queue 1 uses up its time slice at time 19
Queue 2 continues to use its next time slice
                       2 222
Queue 2 completes all its processes at time 23
Queue 3 is empty at time 23

Queue 1 continues to use its next time slice
                            55 3344
Queue 1 uses up its time slice at time 29
Queue 2 continues to use its next time slice
                                    3 44444 44
Queue 2 uses up its time slice at time 37
Queue 3 is empty at time 37

Queue 1 continues to use its next time slice
                                                 55
Queue 1 completes all its processes at time 39
Queue 2 continues to use its next time slice
                                                   4 55555 55
Queue 2 uses up its time slice at time 47
Queue 3 is empty at time 47
Queue 1 is empty at time 47
Queue 2 continues to use its next time slice
                                                          5
Queue 2 completes all its processes at time 48
Processes on queue 3 can now begin, only P5 is there
                                                          55
All queues become empty at time 50
```

```
Summarizing
Q1: 11221 1         33 2244      55 3344           55
Q2:         1111 111         2 222         3 44444 44   4 55555 555
Q3:                                                                55
```

Pid	Burst	Arr	Prior	Wait	TR
1	11	0	4	2	13
2	8	1	2	14	22
3	5	3	1	22	27
4	12	6	5	22	34
5	14	8	3	28	42
Avg.				17.6	27.6

3. Contiguous Memory Allocation.

The following indicates filling up of holes by the tasks. Note that holes do not occupy consecutive memory space and there is a boundary between them composed of occupied memory space. The holes left behind after the allocation and the unfilled requests are painted in yellow and pink respectively. There are 7 possible ways to fill up the holes in an optimal manner.

Alg.	234 hole				321 hole							108 hole			Unfilled		Util.
FF	50	88	96		37	93	35	63	27	49	**17**	68	33	**7**	58		97.6%
BF	88	96	37	**13**	93	63	27	49	68	**21**		50	35	**23**	33	58	94.3%
WF	96	35	27	49	**27**	50	88	37	93	33	**20**	63		**45**	68	58	90.8%
Opt_1	88	63	49	33	**1**	96	37	93	27	68		50	58		35		99.9%
Opt_2	88	96	49	**1**		37	93	63	27	68	33	50	58		35		99.9%
Opt_3	96	37	68	33		50	88	93	63	27		49	58	**1**	35		99.9%
Opt_4	50	88	63	33		96	37	93	27	68		49	58	**1**	35		99.9%
Opt_5	50	88	96			37	93	63	27	68	33	49	58	**1**	35		99.9%
Opt_6	96	37	68	33		88	93	63	27	49	**1**	50	58		35		99.9%
Opt_7	88	37	27	49	33	96	93	63	68	**1**		50	58		35		99.9%

(*e*) If the size of one of the holes is under-reported, utilization can improve if that hole is the first hole. In other words, the size of the first hole is 235 for that case. There are two possible ways to achieve the full utilization. Note that increasing the size of the two other holes cannot lead to improved utilization.

Alg.	235 hole				321 hole					108 hole		Unfilled	Util.
Opt_{e1}	96	63	27	49	88	37	93	68	35	50	58	33	100.0%
Opt_{e2}	88	63	49	35	96	37	93	27	68	50	58	33	100.0%

(*f,g,h*) The following table indicates the maximum usage of the hole of size *S* and the respective number of tasks allocated. There may be more than one possible answers, as long as your collection of tasks can lead to the usage of 488, 552 and 772 respectively, with at most 9 tasks of each type in (*f*). Note that you *cannot* achieve full usage for the second and third cases. Even if we relax the requirement that there can be unlimited number of tasks of each type as in (*g*), the two holes still cannot be completely filled, but the unused space is smaller for both cases. Finally, if we further strengthen the requirement that tasks of all sizes must be included as in (*h*), the utilizations for the incompletely filled second and third cases in (*f*) will be further reduced.

	(*f*) At most 9 tasks each			(*g*) Unlimited number of tasks			(*h*) Strictly 1 to 9 tasks each		
Case	**1**	**2**	**3**	**1**	**2**	**3**	**1**	**2**	**3**
x	22×1/4	26×8	28×9	22×1/4	26×18	28×23/25	22×1	26×5/7/9	28×4/6/8
y	33×2/0	43×8	65×8	33×2/0	43×2	65×2/0	33×2	43×8/5/2	65×9/7/5
z	50×8/8	77×0	74×0	50×8/8	77×0	74×0/1	50×8	77×1/2/3	74×1/2/3
S	488	556	777	488	556	777	488	556	777
Max usage	488	552	772	488	554	774	488	551	771

4. Segmentation.

Memory used: 0 to 999, 1002 to 1247, 1433 to 1553, 2432 to 2735, 3131 to 3475, 3911 to 4342, 4431 to 5207, 5678 to 6208, 6789 to 8191. So there are 8 holes (1000 to 1001, 1248 to 1432, 1554 to 2431, 2736 to 3130, 3476 to 3910, 4343 to 4430, 5208 to 5677, 6209 to 6788) of sizes 2, 185, 878, 395, 435, 88, 470, 580 to be allocated for 7 segments (*you **better draw** three diagrams to show the memory allocation for the three segments with FF, BF, WF algorithms as in Tutorial 8, either horizontally or vertically*).

First-fit allocates S_1, S_6 to 2nd hole, S_0, S_2, S_5 to to 3rd hole, and S_3 to 8th hole. S_4 is shared with S_3 of P_1.
Best-fit allocates S_0 to 4th hole, S_2, S_6 to 5th hole, S_1 to 6th hole, S_5 to 7th hole, and S_3 to 8th hole. S_4 is shared with S_3 of P_1.
Worst-fit allocates S_0, S_1, S_3 to 3rd hole, S_6 to 5th hole, S_5 to 7th hole, and S_2 to 8th hole. S_4 is shared with S_3 of P_1.

Segment table for P_2	*Base*			*Length / Limit*
	(*a*) **FF**	(*b*) **BF**	(*c*) **WF**	
0	1554	2736	1554	212
1	1248	4343	1766	88
2	1766	3476	6209	345
3	6209	6209	1854	511
4	2432	2432	2432	304
5	2111	5208	5208	321
6	1336	3821	3476	72

(*d*) *Translating* logical addresses into physical addresses.

Allocation algorithm for P_2		**FF**	**BF**	**WF**
Logical address	Physical address for P_1	Physical address for P_2		
(0, 19)	1452	1573	2755	1573
(1, 88)	4519	error/invalild	error/invalild	error/invalild
(2, 246)	3377	2012	3722	6455
(3, 304)	error/invalild	6513	6513	2158
(4, 188)	5866	2620	2620	2620
(5, 234)	1236	2345	5442	5442
(6, 33)	3944	1369	3854	3509