

Nama : Nashirudin Baqiy

NIM : 24060119130045

Kelas : A1

Tugas 5

Praktikum Sistem Operasi

I. 5a Pembuatan Thread

1. thread1.c

```
1 #include <stdlib.h>
2 #include <pthread.h>
3 #include <stdio.h>
4 #include <unistd.h>
5
6 // we must make the compiler aware that this program
7 // is to use threads so the thread safe libraries must be used.
8 // To do this we define the _REENTRANT flag
9 #define _REENTRANT
10 //next we create a simple function for out thread
11 void *ThreadRoutine(int number)
12 {
13     while(1) // loop forever
14     {
15         printf("pid %d : thread %d running\n",getpid(), number);
16         sleep(number); // sleep for a time passed as a parameter
17     }
18 }
19 int main(void)
20 {
21     int t;
22     pthread_t tid[5]; // an array to keep track of the threads
23     // now loop through and create 4 threads passing t as the parameter
24     for (t=1; t<5; t++)
25         pthread_create(&tid[t],NULL,(void *)ThreadRoutine, &t);
26
27     // now the parent loops and sleeps for 10
28
29     while(1)
30     {
31         printf("pid %d : parent running\n", getpid());
32         sleep(10);
33     }
34     exit(1);
35 }
```

Output:

```
nashirudin@deen-VirtualBox:~/Tugas5$ ./thread1
pid 3353 : parent running
pid 3353 : thread -82537828 running
pid 3353 : thread -82537828 running
pid 3353 : thread -82537828 running
pid 3353 : thread -82537828 running
pid 3353 : parent running
pid 3353 : parent running
pid 3353 : parent running
pid 3353 : parent running
pid 3353 : parent running
pid 3353 : parent running
pid 3353 : parent running
pid 3353 : parent running
pid 3353 : parent running
pid 3353 : parent running
pid 3353 : parent running
pid 3353 : parent running
pid 3353 : parent running
```

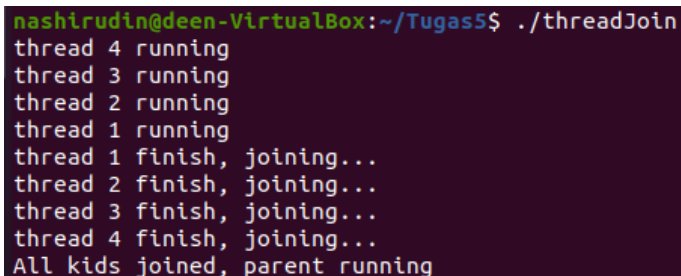
Pada program thread1.c define `_REENTRANT` untuk memberitahu compiler bahwa codenya ada thread. Thread tid didefinisikan `pthread_t`. Dibuat variabel array yang menyimpan track of the threads. Parent dijalankan terlebih dahulu karena traversalnya menunggu return parentnya lalu `sleep(10)`. Kemudian dijalankan traversal `pthread_create` membuat thread dan menjalankan ThreadRoutine sampai 4 threads. ThreadRoutine adalah isi thread.

Terakhir, hanya ada looping terus menerus tiap 10 detik terhadap parent. Pengaruh nilai `sleep` pada program thread1.c yaitu membiarkan parent running dahulu agar dapat menjalankan threads dan konstan `sleep 10` detik terhadap looping parent.

2. threadJoin.c

```
1 #include <pthread.h>
2 #include <stdio.h>
3 #include <unistd.h>
4 #include <stdlib.h>
5
6 #define _REENTRANT
7 //next we create a simple function for out thread
8 void *ThreadRoutine(int number)
9 {
10     printf("thread %d running\n",number);
11     sleep(number); // sleep for a time passed as a parameter
12     printf("thread %d finish, joining... \n",number);
13 }
14 int main(void)
15 {
16     int t;
17     pthread_t tid; // an array to keep track of the threads
18     // now loop through and create 4 threads passing t as the parameter
19     for (t=1; t<5; t++)
20         pthread_create(&tid,NULL,(void *)ThreadRoutine,(int *)t);
21
22     // now the calling process waits for the thread to finish
23     pthread_join(tid,NULL);
24     printf("All kids joined, parent running\n");
25     exit(1);
26 }
```

Output:

A screenshot of a terminal window showing the output of the threadJoin.c program. The prompt is 'nashirudin@deen-VirtualBox:~/Tugas5\$./threadJoin'. The output shows four threads running in parallel, each printing 'thread X running' and 'thread X finish, joining...' before the parent thread prints 'All kids joined, parent running'.

```
nashirudin@deen-VirtualBox:~/Tugas5$ ./threadJoin
thread 4 running
thread 3 running
thread 2 running
thread 1 running
thread 1 finish, joining...
thread 2 finish, joining...
thread 3 finish, joining...
thread 4 finish, joining...
All kids joined, parent running
```

Pada program threadJoin.c define `_REENTRANT` untuk memberitahu compiler bahwa codenya ada thread. Thread `tid` didefinisikan `pthread_t`. Melakukan traversal `pthread_create` membuat thread (child) dan menjalankan `ThreadRoutine` sebanyak 4 kali. `ThreadRoutine` adalah isi thread. `ThreadRoutine` mengandung penjalanan thread, thread manapun menunggu dengan `sleep(number)` agar semua thread dirun sebelum thread finishing. Keempat thread membagi resource yang sama dalam satu proses.

Terakhir, dijalankan `pthread_join`. Tujuan penggunaan `pthread_join` pada program threadJoin.c ini adalah memanggil `tid` (parent) bergabung dengan thread-thread lain penjadwalan terminasinya. Maka `tid` (parent) akan berjalan selama thread lain berjalan dan terminasi bila thread lain terminated. Hal ini agar parent menunggu childnya selesai.

3. threadDetach.c

roundrobin.c	×	thread1.c	×
<pre>1 #include <stdlib.h> 2 #include <pthread.h> 3 #include <stdio.h> 4 #include <unistd.h> 5 #define _REENTRANT 6 //next we create a simple function for our thread 7 void *ThreadRoutine(int number) 8 { 9 int i; 10 for (i=0; i<10; i++) //loop to give the thread something to do 11 { 12 printf("thread %d running %d\n",number,i); 13 sleep(number); // sleep for a time passed as a parameter 14 } 15 } 16 int main(void) 17 { 18 pthread_t tid1,tid2; // create 2 thread id's 19 //now create two threads 20 pthread_create(&tid1,NULL,(void *)ThreadRoutine,(int *)1); 21 pthread_create(&tid2,NULL,(void *)ThreadRoutine,(int *)2); 22 pthread_detach(tid1); //we will now detach thread 1 23 if(pthread_join(tid1,NULL)>0) // now try to join it 24 printf("unable to join thread 1\n"); 25 if(pthread_join(tid2,NULL)>0) // and now join thread 2 26 printf("unable to join thread 2\n"); 27 28 printf("parent finished\n"); 29 exit(1); 30 }</pre>			

Output pthread_detach(tid1):

```
nashirudin@deen-VirtualBox:~/Tugas5$ ./threadDetach
unable to join thread 1
thread 2 running 0
thread 1 running 0
thread 1 running 1
thread 2 running 1
thread 1 running 2
thread 2 running 2
thread 1 running 4
thread 1 running 5
thread 2 running 3
thread 1 running 6
thread 1 running 7
thread 2 running 4
thread 1 running 8
thread 1 running 9
thread 2 running 5
thread 2 running 6
thread 2 running 7
thread 2 running 8
thread 2 running 9
parent finished
```

Output pthread_detach(tid2):

```
nashirudin@deen-VirtualBox:~/Tugas5$ ./threadDetach2
thread 2 running 0
thread 1 running 0
thread 1 running 1
thread 2 running 1
thread 1 running 2
thread 1 running 3
thread 2 running 2
thread 1 running 4
thread 1 running 5
thread 2 running 3
thread 1 running 6
thread 1 running 7
thread 2 running 4
thread 1 running 8
thread 1 running 9
thread 2 running 5
unable to join thread 2
parent finished
```

Pada program threadDetach.c define `_REENTRANT` untuk memberitahu compiler bahwa codenya ada thread. Thread `tid1` `tid2` didefinisikan `pthread_t`. dibuat 2 threads dengan `pthread_create` lalu menjalankan ThreadRoutine yang mengandung traversal running thread sebanyak 10 kali yang index awalnya 0. ThreadRoutine adalah isi thread. Setelah `pthread_create`, dijalankan `pthread_detach` untuk melepas thread. Di sini kita membandingkan `pthread_detach` `tid1` dan `tid2`.

Ketika detach `tid1`, `tid1` dilepas koneksi joinablenya kemudian mengembalikan nilai error saat join di awal. Ketika detach `tid2`, `tid2` dilepas koneksi joinablenya kemudian mengembalikan nilai error join saat `tid1` selesai. Error join saat detach `tid1` di awal karena `tid1` berhasil dilepas duluan sebelum join. Join saat detach `tid2` tidak error karena join telah terjadi dan tidak bisa diganggu oleh detach sehingga baru mengembalikan error join ketika thread lain (`tid1`) terminated.

Fungsi `pthread_detach` pada program threadDetach.c yaitu menandai thread bila terminated maka resource yang dipakai dikembalikan ke sistem secara otomatis. Perbedaan dari detach `tid1` dan `tid2` adalah, detach `tid1` resource tetap berjalan disistem meski `tid1` terminated sehingga `tid2` tetap berjalan sedangkan detach `tid2` berhenti ketika `tid1` terminated padahal membutuhkannya.

4. globaldata.c

```
1 #include <stdio.h>
2 #include <pthread.h>
3 int glob_data = 5 ;
4 void *kidfunc(void *p)
5 {
6     printf ("Kid here. Global data was %d.\n", glob_data) ;
7     glob_data = 15 ;
8     printf ("Kid Again. Global data was now %d.\n", glob_data) ;
9 }
10 int main ( )
11 {
12     pthread_t kid ;
13     pthread_create (&kid, NULL, kidfunc, NULL) ;
14     printf ("Parent here. Global data = %d\n", glob_data) ;
15     glob_data = 10 ; pthread_join (kid, NULL) ;
16     printf ("End of program. Global data = %d\n", glob_data) ;
17 }
```

Output:

```
nashirudin@deen-VirtualBox:~/Tugas5$ ./globaldata
Parent here. Global data = 5
Kid here. Global data was 10.
Kid Again. Global data was now 15.
End of program. Global data = 15
```

Penjelasan globaldata.c, definisi kid sebagai pthread_t. Kemudian pthread_create membuat thread kid namun kidfunc menunggu parent jalan dahulu. Parent mempunyai 5 global data. Kemudian global data ditambah 5 dari kid pertama menjadi 10. Thread kid join ke parent. Barulah dijalankan kidfunc karena parent menunggu kid dahulu. Membuat kid lagi bertambah 5 global data menjadi 15.

Pada percobaan globaldata.c suatu variabel **dapat** diakses oleh semua thread. Buktinya yaitu parent, kid lalu kidnya lagi semua terhitung sebagai global data.

II. 5b Penjadwalan Proses

First come first serve=datang lebih dulu maka lebih dulu dilayani, sjf=shortest job first, pekerjaan paling kecil selesaikan lebih dulu

Roundrobin mendukung multitasking.

1. Bandingkan implementasi program FCFS dengan SJF, mana yang lebih ringkas? Beri penjelasannya.

FCFS

fcfs.c

```
1 #include<stdio.h>
2
3 int main()
4 {
5     int bt[20], wt[20], tat[20], i, n;
6     float wtavg, tatavg;
7     printf("\nEnter the number of processes -- ");
8     scanf("%d", &n);
9     for(i=0;i<n;i++)
10    {
11        printf("\nEnter Burst Time for Process %d -- ", i); scanf("%d",
12            &bt[i]);
13    }
14    wt[0] = wtavg = 0;
15    tat[0] = tatavg = bt[0];
16    for(i=1;i<n;i++)
17    {
18        wt[i] = wt[i-1] +bt[i-1];
19        tat[i] = tat[i-1] +bt[i];
20        wtavg = wtavg + wt[i];
21        tatavg = tatavg + tat[i];
22    }
23    printf("\t PROCESS \tBURST TIME \t WAITING TIME\t TURNAROUND TIME\n");
24    for(i=0;i<n;i++)
25        printf("\n\t P%d \t\t %d \t\t %d \t\t %d", i, bt[i], wt[i],
26            tat[i]);
27    printf("\nAverage Waiting Time -- %f", wtavg/n);
28    printf("\nAverage Turnaround Time -- %f", tatavg/n);
29 }
```

Output:

```
nashirudin@deen-VirtualBox:~/Tugas5/5b$ ./fcfs
Enter the number of processes -- 5
Enter Burst Time for Process 0 -- 2
Enter Burst Time for Process 1 -- 7
Enter Burst Time for Process 2 -- 1
Enter Burst Time for Process 3 -- 5
Enter Burst Time for Process 4 -- 4

```

PROCESS	BURST TIME	WAITING TIME	TURNAROUND TIME
P0	2	0	2
P1	7	2	9
P2	1	9	10
P3	5	10	15
P4	4	15	19

```
Average Waiting Time -- 7.200000
Average Turnaround Time -- 11.000000nashirudin@deen-VirtualBox:~/Tugas5/5b$
```

SJF

sjf.c

```
1 #include<stdio.h>
2
3 int main()
4 {
5     int p[20], bt[20], wt[20], tat[20], i, k, n, temp;
6     float wtavg, tatavg;
7     printf("\nEnter the number of processes -- ");
8     scanf("%d", &n);
9     for(i=0;i<n;i++)
10    {
11        p[i]=i;
12        printf("Enter Burst Time for Process %d -- ", i);
13        scanf("%d", &bt[i]);
14    }
15    /* mengurutkan proses dengan urutan burst time terkecil */
16    for(i=0;i<n;i++)
17        for(k=i+1;k<n;k++)
18            if(bt[i]>bt[k])
19            {
20                temp=bt[i];
21                bt[i]=bt[k];
22                bt[k]=temp;
23                temp=p[i];
24                p[i]=p[k];
25                p[k]=temp;
26            }
27    wt[0] = wtavg = 0;
28    tat[0] = tatavg = bt[0];
29    for(i=1;i<n;i++)
30    {
31        wt[i] = wt[i-1] +bt[i-1];
32        tat[i] = tat[i-1] +bt[i];
33        wtavg = wtavg + wt[i];
34        tatavg = tatavg + tat[i];
35    }
36    printf("\n\tPROCESS \tBURST TIME \t WAITING TIME\t TURNAROUND TIME\n");
37    for(i=0;i<n;i++)
38        printf("\n\t P%d \t\t %d \t\t %d \t\t %d", p[i], bt[i], wt[i],
39        tat[i]);
40    printf("\nAverage Waiting Time -- %f", wtavg/n);
41    printf("\nAverage Turnaround Time -- %f", tatavg/n);
42 }
```

Output:

```
Average Turnaround Time -- 11.000000nashirudin@deen-VirtualBox:~/Tugas5/5b$ ./sjf
Enter the number of processes -- 5
Enter Burst Time for Process 0 -- 2
Enter Burst Time for Process 1 -- 7
Enter Burst Time for Process 2 -- 1
Enter Burst Time for Process 3 -- 5
Enter Burst Time for Process 4 -- 4

```

PROCESS	BURST TIME	WAITING TIME	TURNAROUND TIME
P2	1	0	1
P0	2	1	3
P4	4	3	7
P3	5	7	12
P1	7	12	19

```
Average Waiting Time -- 4.600000
Average Turnaround Time -- 8.400000nashirudin@deen-VirtualBox:~/Tugas5/5b$
```

Dapat dilihat perbandingannya, average waiting time lebih rendah SJF dan average turnaround time lebih rendah sjf. Dapat disimpulkan SJF lebih ringkas karena SJF mengefisienkan pengambilan process dari yang terpendek lebih dahulu. Awal burst time pendek dan beban burst time tertinggi di akhir.

2. Dengan jumlah proses dan nilai burst time yang sama pada percobaan FCFS, SJF, dan RR, bandingkan nilai average waiting-time dan average turn-around time, algoritma mana yang terbaik?

RR

roundrobin.c

```
1 #include<stdio.h>
2
3 main()
4 {
5     int i,j,n,bu[10],wa[10],tat[10],t,ct[10],max;
6     float awt=0,att=0,temp=0;
7     printf("Enter the no of processes -- ");
8     scanf("%d",&n);
9     for(i=0;i<n;i++)
10    {
11        printf("\nEnter Burst Time for process %d -- ", i+1);
12        scanf("%d",&bu[i]);
13        ct[i]=bu[i];
14    }
15    printf("\nEnter the size of time slice -- ");
16    scanf("%d",&t);
17    max=bu[0];
18    for(i=1;i<n;i++) /* temukan proses dengan burst time terbesar */
19    if(max<bu[i])
20        max=bu[i];
21    for(j=0;j<(max/t)+1;j++)
22        for(i=0;i<n;i++) /* iterasi pada setiap proses */
23            if(bu[i]!=0)
24                if(bu[i]<=t) /*jika kurang dari time slice*/
25                {
26                    tat[i]=temp+bu[i]; /* catat ta-time */
27                    temp=temp+bu[i];
28                    bu[i]=0;
29                }
30            else /*jika lebih besar dari time slice*/
31            {
32                bu[i]=bu[i]-t;
33                temp=temp+t;
34            }
35    for(i=0;i<n;i++)
36    {
37        wa[i]=tat[i]-ct[i]; /* catat waiting time tiap proses*/
38        att+=tat[i]; /* catat total turn around time */
39        awt+=wa[i]; /* catat total waiting time */
40    }
41    printf("\nThe Average Turnaround time is -- %f",att/n);
42    printf("\nThe Average Waiting time is -- %f ",awt/n);
43    printf("\n\tPROCESS\t BURST TIME \t WAITING TIME\tTURNAROUND TIME\n");
44    for(i=0;i<n;i++)
45        printf("\t%d \t %d \t\t %d \t\t %d \n",i+1,ct[i],wa[i],tat[i]);
46
47 }
```

Output roundrobin (tidak urut dari terkecil) time slice 2:

```
nashirudin@deen-VirtualBox:~/Tugas5/5b$ ./roundrobin
Enter the no of processes -- 5

Enter Burst Time for process 1 -- 2
Enter Burst Time for process 2 -- 7
Enter Burst Time for process 3 -- 1
Enter Burst Time for process 4 -- 5
Enter Burst Time for process 5 -- 4

Enter the size of time slice -- 2

The Average Turnaround time is -- 11.800000
The Average Waiting time is -- 8.000000
```

PROCESS	BURST TIME	WAITING TIME	TURNAROUND TIME
1	2	0	2
2	7	12	19
3	1	4	5
4	5	13	18
5	4	11	15

Output roundrobin time slice 6:

```
nashirudin@deen-VirtualBox:~/Tugas5/5b$ ./roundrobin
Enter the no of processes -- 5

Enter Burst Time for process 1 -- 2
Enter Burst Time for process 2 -- 7
Enter Burst Time for process 3 -- 1
Enter Burst Time for process 4 -- 5
Enter Burst Time for process 5 -- 4

Enter the size of time slice -- 6

The Average Turnaround time is -- 12.400000
The Average Waiting time is -- 8.600000
```

PROCESS	BURST TIME	WAITING TIME	TURNAROUND TIME
1	2	0	2
2	7	12	19
3	1	8	9
4	5	9	14
5	4	14	18

Output roundrobin (urut dari terkecil) time slice 2:

```
nashirudin@deen-VirtualBox:~/Tugas5/5b$ ./roundrobin
Enter the no of processes -- 5

Enter Burst Time for process 1 -- 1
Enter Burst Time for process 2 -- 2
Enter Burst Time for process 3 -- 4
Enter Burst Time for process 4 -- 5
Enter Burst Time for process 5 -- 7

Enter the size of time slice -- 2

The Average Turnaround time is -- 10.000000
The Average Waiting time is -- 6.200000


| PROCESS | BURST TIME | WAITING TIME | TURNAROUND TIME |
|---------|------------|--------------|-----------------|
| 1       | 1          | 0            | 1               |
| 2       | 2          | 1            | 3               |
| 3       | 4          | 7            | 11              |
| 4       | 5          | 11           | 16              |
| 5       | 7          | 12           | 19              |


```

RoundRobin akan semakin rendah average turnaround dan waiting timenya apabila menggunakan konsep SJF yaitu dari terpendek. Time slice juga berpengaruh, time slice lebih baik melihat burst time tiap proses agar time slice memiliki waiting time sesedikit mungkin tiap proses tiap round.

Output FCFS:

```
nashirudin@deen-VirtualBox:~/Tugas5/5b$ ./fcfs
Enter the number of processes -- 5
Enter Burst Time for Process 0 -- 2
Enter Burst Time for Process 1 -- 7
Enter Burst Time for Process 2 -- 1
Enter Burst Time for Process 3 -- 5
Enter Burst Time for Process 4 -- 4


| PROCESS | BURST TIME | WAITING TIME | TURNAROUND TIME |
|---------|------------|--------------|-----------------|
| P0      | 2          | 0            | 2               |
| P1      | 7          | 2            | 9               |
| P2      | 1          | 9            | 10              |
| P3      | 5          | 10           | 15              |
| P4      | 4          | 15           | 19              |


Average Waiting Time -- 7.200000
Average Turnaround Time -- 11.000000nashirudin@deen-VirtualBox:~/Tugas5/5b$
```

Output SJF:

```
Average Turnaround Time -- 11.000000nashlrudin@deen-VirtualBox:~/Tugas5/5b$ ./sjf
Enter the number of processes -- 5
Enter Burst Time for Process 0 -- 2
Enter Burst Time for Process 1 -- 7
Enter Burst Time for Process 2 -- 1
Enter Burst Time for Process 3 -- 5
Enter Burst Time for Process 4 -- 4

      PROCESS      BURST TIME      WAITING TIME      TURNAROUND TIME
      P2           1           0           1
      P0           2           1           3
      P4           4           3           7
      P3           5           7          12
      P1           7          12          19
Average Waiting Time -- 4.600000
Average Turnaround Time -- 8.400000nashlrudin@deen-VirtualBox:~/Tugas5/5b$
```

Berdasarkan average waiting time dan average turnaround time menjadikan SJF sebagai algoritma yang terbaik. Urut dari terkecil sangat mempengaruhi waiting time, sehingga juga mendapatkan turnaround time yang minimum. Dapat disimpulkan bahwa konsep algoritma SJF yang urut dari terkecil pemenang terbaik.

3. Berdasarkan hasil pengamatan terhadap ketiga algoritma penjadwalan, apakah algoritma RR memiliki kelebihan dibandingkan FCFS dan SJF? Beri penjelasannya (silahkan merujuk buku referensi)

Kelebihan dari algoritma RR (RoundRobin) dibandingkan FCFS dan SJF adalah RR dapat multitasking untuk mempersingkat waiting time apabila burst time antar proses memiliki kesenjangan yang rendah dan memiliki time slice serendah mungkin dari rata-rata burst time yang dapat menyelesaikan banyak proses dalam ronde sesedikit mungkin.