VO 182.711

Prüfung Betriebssysteme

19. June 2018

KNr. MNr. Zuname, Vorname

Ges. (100)	1.)(30)	2.)(20)	3.)(25)	4.)(25)	Zusatzblätter:

Bitte verwenden Sie nur dokumentenechtes Schreibmaterial !

1 Synchronisation (30)

German version

Das Leser/Schreiber-Problem ist wie folgt definiert: Ein Datenbereich wird von mehreren Prozessen geteilt. Der Datenbereich könnte eine Datei, ein Speicherblock im Hauptspeicher oder eine Reihe von Prozessorregister sein. Eine Teilmenge der Prozesse greift nur lesend auf den Datenbereich zu (Leser) und ein andere Teilmenge der Prozesse greift nur schreibend auf den Datenbereich zu (Schreiber). Folgende Bedienungen müssen bei jedem Datenzugriff gewährleistet sein:

1. Eine beliebige Anzahl an Prozessen kann gleichzeitig vom Datenbereich lesen
2. Zu jeder Zeit darf maximal ein Prozess auf den Datenbereich schreibend zugreifen
3. Während ein Schreiber auf den Datenbereich zugreift darf kein Prozess vom Datenbereich lesen
4. Schreiber dürfen nicht verhungern: Kein Leser darf auf den Datenbereich zugreifen sobald zumindest ein Schreiber bekanntgegeben hat, dass dieser auf den Datenbereich schreiben möchte.

Implementieren Sie eine Lösung die, welche die im Initialisierungscode gegebene Semaphoren und Variablen verwendet. Zum Lesen und Schreiben auf den Datenbereich sollen die Funktionen *read*() bzw. *write*() verwendet werden.

English version

The readers/writers problem is defined as follows: There is a data area shared among a number of processes. The data area could be a file, a block of main memory, or even a bank of processor registers. There are a number of processes that only read the data area (readers) and a number of processes that only write the data area (writers). The conditions that must be satisfied are as follows:

1. Any number of readers may simultaneously read the file
2. Only one writer at the time may write to the file
3. If a writer is writing to the file, no reader may read it
4. writers do not starve: no new readers are allowed access to the data once at least one writer has declared a desire to write.

Implement a solution using the semaphores reported in the initialisation code. The instructions for reading and writing are *read*() and *write*() respectively.

Code for Initialization

```
semaphore     x = 1;
semaphore     y = 1;
semaphore     z = 1;
semaphore wsem = 1; //Writing semaphors
semaphore rsem = 1;  //Reading semaphors
int          rc = 0;
int          wc = 0;
```

Code for Process Reader

```
void reader (void){
        while (true) {
        // Complete the code



        }
}
```

Code for Process Writer

```
void writer (void){
        while (true) {
        // Complete the code



        }
}
```

2 Uniprocessor Scheduling (20)

German version

Gegeben sei folgendes Taskset:

Process	Arrival Time	Service Time
A	0	3
B	1	4
C	2	8
D	8	5
E	12	4

Schedulen Sie die Tasks im untenstehenden Raster (wie am Beispiel FCFS gezeigt) für die folgenden Scheduling Algorithmen:

- Round robin (Zeitscheibenlänge q = 2 und q = 3)
- Shortest process next (SPN)
- Shortest remaining time (STR)
- Highest response ratio next (HRRN)

- In SRT: Behandeln Sie Tasks in alphabetischer Reihenfolge, wenn Tasks dieselbe verbleibende Service-Zeit haben.
- In RR (zum Beispiel q=1): Wählen Sie den neuen Task, wenn ein neuer Task ankommt und die Zeitscheibe für den vorherigen Task endet.

English version

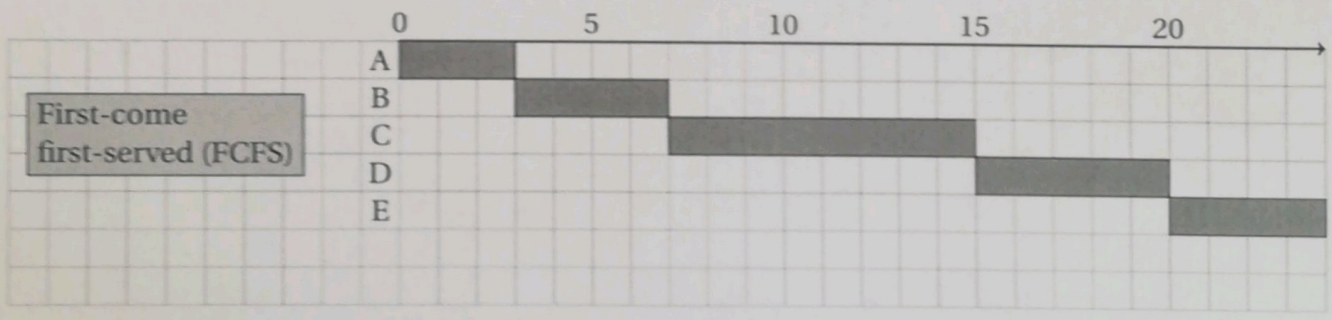
Consider the following set of processes:

Process	Arrival Time	Service Time
A	0	3
B	1	4
C	2	8
D	8	5
E	12	4

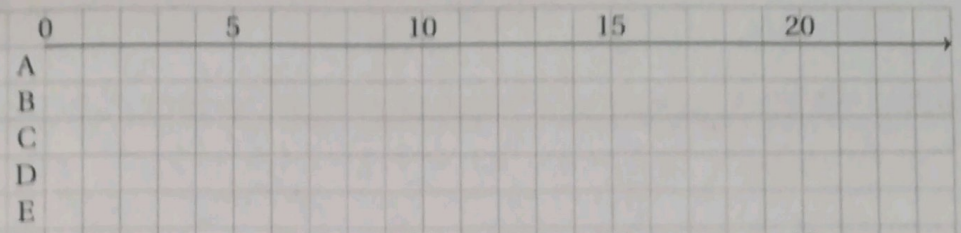
Perform the scheduling analysis (as done for the FCFS as an example) using the grid below for the following scheduling algorithms:

- Round robin (using time quanta of q=2 and q=3)
- Shortest process next (SPN)
- Shortest remaining time (STR)
- Highest response ratio next (HRRN)

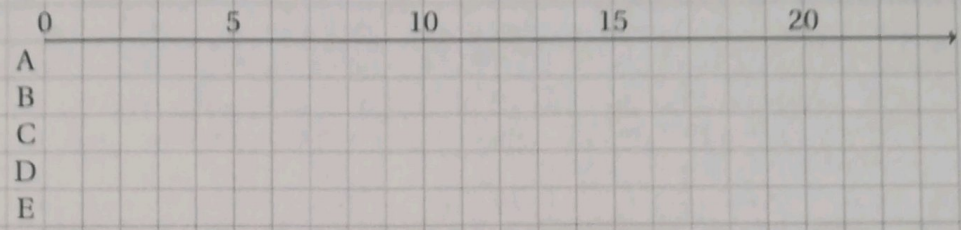
- In SRT if tasks have equal remaining service time we handle them in an alphabetical order.
- In the case of RR (for example q=1), when a new task arrives and the time quanta for the previous task finishes, the new task is chosen.



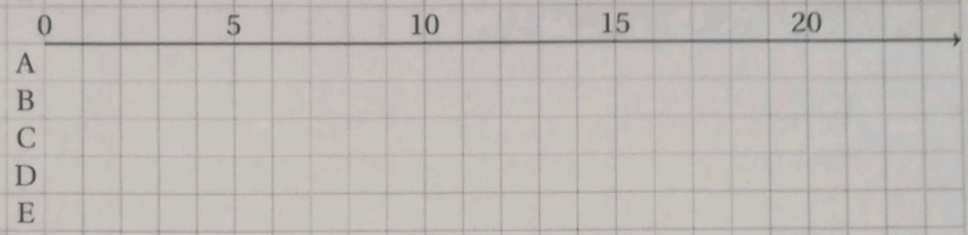
Round-robin (RR), q=2



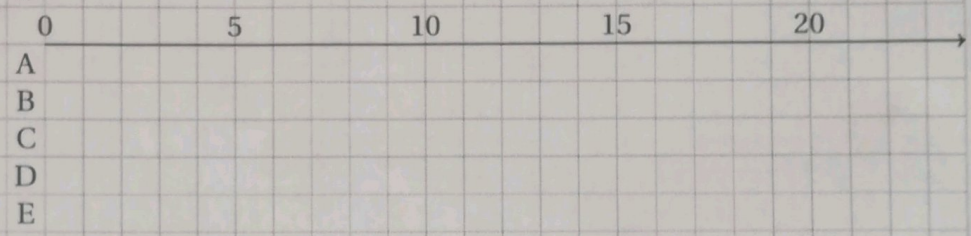
Round-robin (RR), q=3



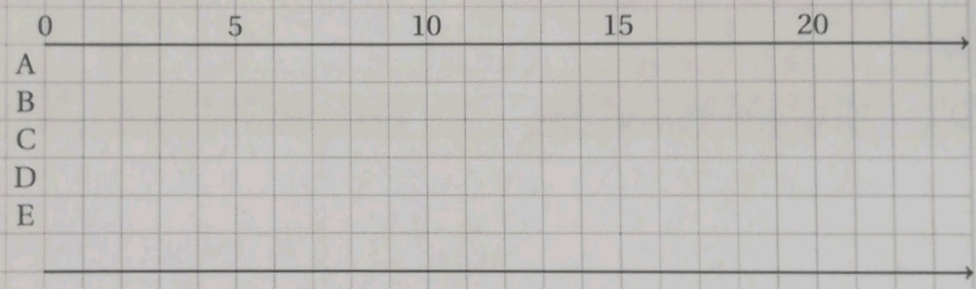
Shortest process next (SPN)



Shortest remaining time (STR)



Highest response ratio next (HRRN)



3 Deadlock (25)

German version

In den folgenden Codestücken konkurrieren 3 Prozesse P0 bis P2 um 6 Ressourcen A bis F. Zeigen Sie die Möglichkeit eines Deadlocks in dieser Implementierung mittels "Resource Allocation Graph" (siehe "Example").

English version

In the code below, three processes are competing for six resources labeled A to F. Show the possibility of a deadlock in this implementation, using a resource allocation graph. See Fig. for examples of resource allocation graphs.

Process P0

```
void P0 ()
{
    //
    while (true)
    {
        //
        get(A);
        get(E);
        get(B);
        criticalS1();
        release(B);
        release(E);
        release(A);
    }
    //
}
```

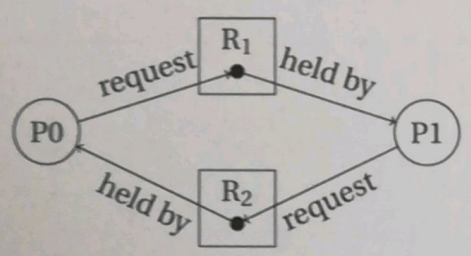
Process P1

```
void P1 ()
{
    //
    while (true)
    {
        get(C);
        get(D);
        get(B);
        get(A);
        criticalS2();
        release(A);
        release(B);
        release(D);
        release(C)
    }
}
```

Process P2

```
void P2 ()
{
    while (true)
    {
        get (D);
        get (B);
        get (C);
        get (E);
        criticalS3();
        release(E);
        release(C);
        release(B);
        release(D);
    }
}
```

Example



Resource Allocation Graph

German version

Ändern Sie die Reihenfolge der Ressourcen-Anforderungen *innerhalb eines Prozesses* um jede Möglichkeit eines Deadlocks zu verhindern.

English version

Modify the order of some of the get requests to prevent the possibility of any deadlock. You cannot move requests across procedures, only change the order inside each procedure.

Process P0

```
void P0 ()
{
    //
    while (true)
    {
        //
        get(  );
        get(  );
        get(  );
        criticalS1();
        release(  );
        release(  );
        release(  );
    }
    //
}
```

Process P1

```
void P1 ()
{
    //
    while (true)
    {
        get(  );
        get(  );
        get(  );
        get(  );
        criticalS2();
        release(  );
        release(  );
        release(  );
        release(  )
    }
}
```

Process P2

```
void P2 ()
{
    while (true)
    {
        get (  );
        get (  );
        get (  );
        get (  );
        criticalS3();
        release(  );
        release(  );
        release(  );
        release(  );

    }
}
```

German version

Gegeben sei folgender Zustand für den "Banker's Algorithm". Snapshot zum Zeitpunkt T0:

Verfügbare Ressourcen

A	B	C	D
6	3	1	9

Process	Aktuell alloziert				Max. Nachfrage				Need matrix			
	A	B	C	D	A	B	C	D	A	B	C	D
P0	9	0	4	0	9	5	5	5				
P1	0	0	0	0	2	2	3	3				
P2	0	1	0	0	7	5	4	4				
P3	1	0	2	0	3	3	3	2				
P4	0	2	1	0	5	2	2	1				
P5	0	0	1	0	3	2	2	4				

1. Berechnen Sie die *Need matrix*. Ist der aktuelle Zustand sicher?

   ☐ Ja. Zeigen Sie, dass der aktuelle Zustand sicher ist indem Sie eine Deadlock freie Reihenfolge zum Ausführen der Processe finden. Berechnen Sie auch die Änderungen der verfügbaren Resourcen wenn ein Prozess terminiert.

   Terminierungsreihenfolge der Prozesse — **Verfügbare Resourcen**

Prozess	A	B	C	D

   ☐ Nein. Wieso ?

2. Gegeben sei die Anforderung (1,3,1,2) des Prozesses P3. Sollte dieser Anforderung stattgegeben werden?

   ☐ Ja. (Begründen anhand der Tabelle.)

   ☐ Nein. (Begründen anhand der Tabelle.)

Verfügbare Ressourcen

A	B	C	D

Process	Aktuell alloziert				Max. Nachfrage				Need matrix			
	A	B	C	D	A	B	C	D	A	B	C	D
P0												
P1												
P2												
P3												
P4												
P5												

Terminierungsreihenfolge der Prozesse — **Verfügbare Resourcen**

Process	A	B	C	D

English version

Given the following state for the Banker's Algorithm. Snapshot at time T0:

Resources available

A	B	C	D
6	3	1	9

Process	Current alloc.				Max. demand				Need matrix			
	A	B	C	D	A	B	C	D	A	B	C	D
P0	9	0	4	0	9	5	5	5				
P1	0	0	0	0	2	2	3	3				
P2	0	1	0	0	7	5	4	4				
P3	1	0	2	0	3	3	3	2				
P4	0	2	1	0	5	2	2	1				
P5	0	0	1	0	3	2	2	4				

1. Calculate the Need matrix. Is the current state safe ?

   ☐ Yes. Show that the current state is safe, that is, show a safe sequence of processes. In addition, show how the Available (working array) changes as each process terminates.

Sequence of Process Termination — **Available Working Array**

Process	A	B	C	D

   ☐ No. Why ?

2. Given the request (1,3,1,2) from Process P3. Should this request be granted ?

   ☐ Yes. (justify using the tables below)

   ☐ No. (justify using the tables below)

Resources available

A	B	C	D

Process	Current alloc.				Max. demand				Need matrix			
	A	B	C	D	A	B	C	D	A	B	C	D
P0												
P1												
P2												
P3												
P4												
P5												

Sequence of Process Termination — **Available Working Array**

Process	A	B	C	D

4 Page Replacement (25)

German version

Ein Prozess referenziert 5 Pages, A, B, C, D, E, in folgender Reihenfolge:

D C D C E E E E E B C B A E B D

Annahme: fixed frame allocation (fixed resident set size) und einen leeren Arbeitsspeicher der Größe von 4 Frames. Benutzen Sie die folgenden 4 Tabellen, um das Verhalten der Page Replacement Policies OPT, LRU and Clock, zu vergleichen. Bitte markeiren Sie in der letzten Zeile der Tabellen auftretende Page Faults *nachdem die Frames des Arbeitsspeichers initial gefüllt sind.*

English version

A process references 5 pages, A, B, C, D, E, in the following order:

D C D C E E E E E B C B A E B D

We assume a fixed frame allocation (fixed resident set size) for this process starting with an empty main memory of four frames. Using the tables below compare the behaviour of these four page replacement policies: OPT, LRU and Clock. Please report in the last row the page fault occurring *after the frame allocation is initially filled.*

OPT-Policy

	D	C	D	C	E	E	E	E	E	B	C	B	A	E	B	D
0																
1																
2																
3																
PF																

LRU-Policy

	D	C	D	C	E	E	E	E	E	B	C	B	A	E	B	D
0																
1																
2																
3																
PF																

Clock-Policy

	D	C	D	C	E	E	E	E	E	B	C	B	A	E	B	D
0																
1																
2																
3																
PF																