CHAPTER 5

CONCURRENCY: MUTUAL EXCLUSION AND SYNCHRONIZATION

HENRY GIDUDU

- Recap
- Exercise Questions

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Faculty of Engineering, Built Environment and Information Technology
Fakulteit Ingenieurswese, Bou-omgewing en Inligtingtegnologie / Lefapha la Boetšenere, Tikologo ya Kago le Theknolotši ya Tshedimošo

Make today matter
www.up.ac.za

RECAP
CONCURRENCY ARISES IN THREE DIFFERENT CONTEXTS:

Multiple Applications

- Invented to allow processing time to be shared among active applications

Structured Applications

- Extension of modular design and structured programming

Operating System Structure

- OS themselves implemented as a set of processes or threads

DIFFICULTIES OF CONCURRENCY

- Sharing of global resources
- Difficult for the OS to manage the allocation of resources optimally
- Difficult to locate programming errors as results are not deterministic and reproducible

RECAP
SOME KEY TERMS RELATED TO CONCURRENCY

atomic operation	A function or action implemented as a sequence of one or more instructions that appears to be indivisible; that is, no other process can see an intermediate state or interrupt the operation. The sequence of instruction is guaranteed to execute as a group, or not execute at all, having no visible effect on system state. Atomicity guarantees isolation from concurrent processes.
critical section	A section of code within a process that requires access to shared resources and that must not be executed while another process is in a corresponding section of code.
deadlock	A situation in which two or more processes are unable to proceed because each is waiting for one of the others to do something.
livelock	A situation in which two or more processes continuously change their states in response to changes in the other process(es) without doing any useful work.
mutual exclusion	The requirement that when one process is in a critical section that accesses shared resources, no other process may be in a critical section that accesses any of those shared resources.
race condition	A situation in which multiple threads or processes read and write a shared data item and the final result depends on the relative timing of their execution.
starvation	A situation in which a runnable process is overlooked indefinitely by the scheduler; although it is able to proceed, it is never chosen.

RECAP
MUTUAL EXCLUSION: APPROACHES

Software Approaches

- Dekker's Algorithm
- Peterson's Algorithm

Hardware support Approaches

- Interrupt disabling
- Special machine Instruction
  - Compare and swap instruction
  - Exchange instruction

RECAP
COUNTING AND BINARY SEMAPHORES

A variable that has an integer value upon which only three operations are defined:

- There is no way to inspect or manipulate semaphores other than these three operations

1) A semaphore may be initialized to a nonnegative integer value
2) The semWait operation decrements the semaphore value
3) The semSignal operation increments the semaphore value

RECAP
MONITORS

- Programming language construct that provides equivalent functionality to that of semaphores and is easier to control
- Implemented in a number of programming languages
  - Concurrent Pascal, Pascal-Plus, Modula-2, Modula-3, Java
- Has also been implemented as a program library Software module consisting of one or more procedures, an initialization sequence, and local data

RECAP
SYNCHRONIZATION

- A monitor supports synchronization by the use of **condition variables** that are contained within the monitor and accessible only within the monitor
  - Condition variables are a special data type in monitors which are operated on by two functions:
    - `cwait(c)`: suspend execution of the calling process on condition `c`
    - `csignal(c)`: resume execution of some process blocked after a `cwait` on the same condition

RECAP
PRODUCER/CONSUMER PROBLEM

General Statement:

- One or more producers are generating data and placing these in a buffer
- A single consumer is taking items out of the buffer one at a time
- Only one producer or consumer may access the buffer at any one time

The Problem:

Ensure that the producer won't try to add data into the buffer if its full, and that the consumer won't try to remove data from an empty buffer

RECAP
MESSAGE PASSING

- The actual function is normally provided in the form of a pair of primitives:

```
send (destination, message)
receive (source, message)
```

- A process sends information in the form of a *message* to another process designated by a *destination*
- A process receives information by executing the `receive` primitive, indicating the *source* and the *message*

RECAP
READERS/WRITERS PROBLEM

- A data area is shared among many processes
  - Some processes only read the data area, (readers) and some only write to the data area (writers)
- Conditions that must be satisfied:
  - Any number of readers may simultaneously read the file
  - Only one writer at a time may write to the file
  - If a writer is writing to the file, no reader may read it

CHAPTER 5: QUESTION ONE

Consider the following processes P1 and P2 that update the value of the shared variables, x and y, as follows:

```
Process P1 :
(  performs the operations:
       x := x * y
       y ++
)
LOAD R1, X
LOAD R2, Y
MUL R1, R2
STORE X, R1
INC R2
STORE Y, R2
```

```
Process P2 :
(  performs the operations:
       x ++
       y := x * y
)
LOAD R3, X
INC R3
LOAD R4, Y
MUL R4, R3
STORE X, R3
STORE Y, R4
```

CHAPTER 5: QUESTION ONE

Assume that the initial values of x and y are 5 and 3 respectively. P1 enters the system first and so it is required that the output is equivalent to a serial execution of P1 followed by P2. The scheduler in the uniprocessor system implements a pseudo-parallel execution of these two concurrent processes by interleaving their instructions without restricting the order of the interleaving.

CHAPTER 5: SOLUTION (1a)

a) If the processes P1 and P2 had executed serially, what would the values of x and y have been after the execution of both processes?

- Serial execution means a process executes to completion before the next.
- In this Scenario process P1 executes first

Process	Statement	R1	R2	R3	R4	X	Y
		-	-	-	-	5	3

Faculty of Engineering, Built Environment and Information Technology
Fakulteit Ingenieurswese, Bou-omgewing en Inligtingtegnologie / Lefapha la Boetšenere, Tikologo ya Kago le Theknolotši ya Tshedimošo

CHAPTER 5: SOLUTION (1a)

Process	Statement	R1	R2	R3	R4	X	Y
		-	-	-	-	5	3
1	LOAD R1, X	5	-	-	-	5	3
1	LOAD R2, Y	5	3	-	-	5	3
1	MUL R1, R2	5	3	-	-	5	3
1	STORE X, R1	15	3	-	-	15	3
1	INC R2	15	4	-	-	15	3
1	STORE Y, R2	15	4	-	-	15	4

P1
LOAD R1, X
LOAD R2, Y
MUL R1, R2 (X = X*Y)
STORE X, R1
INC R2 (Y = Y +1)
STORE Y, R2

After executing P1, x = 15, y = 4

CHAPTER 5: SOLUTION (1a)

Process	Statement	R1	R2	R3	R4	X	Y
		15	4	-	-	15	4
2	LOAD R3, X	15	4	15	-	15	4
2	INC R3	15	4	16	-	15	4
2	LOAD R4, Y	15	4	16	4	15	4
2	MUL R4, R3	15	4	16	64	15	4
2	STORE X, R3	15	4	16	64	16	64
2	STORE Y, R4	15	4	16	64	16	64

P2

LOAD R3, X
INC R3 (x =x +1)
LOAD R4, Y
MUL R4, R3 (Y = Y*X)
STORE X, R3
STORE Y, R4

At the end of a serial schedule, the values of *x* and *y* are 16 and 64 respectively

CHAPTER 5: SOLUTION (1b)

Write an interleaved concurrent schedule that gives the same output as a serial schedule.

- ➢ Switching of the processes back and forth during execution. (Interleaving)
- ➢ Each Process are given a time slice to process.

P1
LOAD R1, X
LOAD R2, Y
MUL R1, R2
STORE X, R1
INC R2
STORE Y, R2

P2
LOAD R3, X
INC R3
LOAD R4, Y
MUL R4, R3
STORE X, R3
STORE Y, R4

CHAPTER 5: SOLUTION (1b)

Process	Statement	R1	R2	R3	R4	X	Y
		-	-	-	-	5	3
1	LOAD R1, X	5	-	-	-	5	3
1	LOAD R2,Y	5	3	-	-	5	3
1	MUL R1, R2	15	3	-	-	5	3
1	STORE X, R1	15	3	-	-	15	3
2	LOAD R3, X	15	3	15	-	15	3
2	INC R3	15	3	16	-	15	3

P1
LOAD R1, X
LOAD R2,Y
MUL R1, R2 (X=X*Y)
STORE X, R1
INC R2
STORE Y, R2

P2
LOAD R3, X
INC R3
LOAD R4,Y
MUL R4, R3 (Y=Y*X)
STORE X, R3
STORE Y, R4

Faculty of Engineering, Built Environment and Information Technology
Fakulteit Ingenieurswese, Bou-omgewing en Inligtingtegnologie / Lefapha la Boetšenere, Tikologo ya Kago le Theknolotši ya Tshedimošo
Make today matter
www.up.ac.za

CHAPTER 5: SOLUTION (1b)

Process	Statement	R1	R2	R3	R4	X	Y
		15	3	16	-	15	3
1	INC R2	15	4	16	-	15	3
1	STORE Y, R2	15	4	16	-	15	4
2	LOAD R4, Y	15	4	16	4	15	4
2	MUL R4, R3	15	4	16	64	15	4
2	STORE X, R3	15	4	16	64	16	4
2	STORE Y, R4	15	4	16	64	16	64

P1
LOAD R1, X
LOAD R2, Y
MUL R1, R2 (X=X*Y)
STORE X, R1
INC R2
STORE Y, R2

P2
LOAD R3, X
INC R3
LOAD R4, Y
MUL R4, R3 (Y=Y*X)
STORE X, R3
STORE Y, R4

CHAPTER 5: SOLUTION (1c)

Write an interleaved concurrent schedule that gives an output that is different from that of a serial schedule.

Process	Statement	R1	R2	R3	R4	X	Y
		-	-	-	-	5	3
1	LOAD R1, X	5	-	-	-	5	3
1	LOAD R2, Y	5	3	-	-	5	3
1	MUL R1, R2	15	3	-	-	5	3
2	LOAD R3, X	15	3	5	-	5	3
2	INC R3	15	3	6	-	5	3
2	LOAD R4, Y	15	3	6	3	5	3

P1
LOAD R1, X
LOAD R2, Y
MUL R1, R2 (X=X*Y)
STORE X, R1
INC R2
STORE Y, R2

P2
LOAD R3, X
INC R3
LOAD R4, Y
MUL R4, R3 (Y=Y*X)
STORE X, R3
STORE Y, R4

Faculty of Engineering, Built Environment and Information Technology
Fakulteit Ingenieurswese, Bou-omgewing en Inligtingtegnologie / Lefapha la Boetšenere, Tikologo ya Kago le Theknolotši ya Tshedimošo

CHAPTER 5: SOLUTION (1c)

Process	Statement	R1	R2	R3	R4	X	Y
		15	3	6	3	5	3
1	STORE X, R1	15	3	6	3	15	3
1	INC R2	15	4	6	3	15	3
1	STORE Y, R2	15	4	6	3	15	4
2	MUL R4, R3	15	4	6	24	15	4
2	STORE X, R3	15	3	6	24	6	4
2	STORE Y, R4	15	3	6	24	6	24

P1
LOAD R1, X
LOAD R2, Y
MUL R1, R2 (X=X*Y)
STORE X, R1
INC R2
STORE Y, R2

P2
LOAD R3, X
INC R3
LOAD R4, Y
MUL R4, R3 (Y=Y*X)
STORE X, R3
STORE Y, R4

At the end of the schedule, the values of *x* and *y* are 6 and 24 respectively

CHAPTER 5: QUESTION TWO

The algorithm below is Dekker's solution to the mutual exclusion problem for two processes, P1 and P2 in pseudocode. Outline an informal argument (reasoning) which shows that:

a. One process at most is inside its critical region at a time.
b. If both processes are trying to enter their critical regions simultaneously, a decision will be made within a finite time as to which one should be permitted to do so; and
c. If a process is stopped outside its critical region, this cannot influence the progress of the other process.

```
var outside1, outside2: boolean; turn: 1. .2;
begin
  outside1:= true; outside2:= true; turn:= 1;
  cobegin
  "P1" repeat
           label enter
           begin
             repeat
               outside1:= false;
               repeat
                 if outside2 then exit enter;
               until turn = 2;
               outside1:= true;
               repeat until turn = 1;
             forever
           end
           P1 inside;
           turn:= 2; outside1:= true;
           P1 outside;
         forever

  "P2" repeat
           label enter
           begin
             repeat
               outside2:= false;
               repeat
                 if outside1 then exit enter;
               until turn = 1;
               outside2:= true;
               repeat until turn = 2;
             forever
           end
           P2 inside;
           turn:= 1; outside2:= true;
           P2 outside;
         forever
  coend
end
```

CHAPTER 5: QUESTION TWO

```
boolean flag [2];
int turn;
void P0()
{
     while (true) {
          flag [0] = true;
          while (flag [1]) {
               if (turn == 1) {
                    flag [0] = false;
                    while (turn == 1) /* do nothing
*/;
                    flag [0] = true;
               }
          }
          /* critical section  */;
          turn = 1;
          flag [0] = false;
          /* remainder   */;
     }
}
void P1( )
{
     while (true) {
          flag [1] = true;
          while (flag [0]) {
               if (turn == 0) {
                    flag [1] = false;
                    while (turn == 0) /* do nothing
*/;
                    flag [1] = true;
               }
          }
          /* critical section   */;
          turn = 0;
          flag [1] = false;
          /* remainder   */;
     }
}
void main ()
{
     flag [0] = false;
     flag [1] = false;
     turn = 1;
     parbegin (P0, P1);
}
```

Figure 5.2 Dekker's Algorithm

```
var outside1, outside2: boolean; turn: 1. .2;
begin
  outside1:= true; outside2:= true; turn:= 1;
  cobegin
  "P1" repeat
          label enter
          begin
            repeat
              outside1:= false;
              repeat
                if outside2 then exit enter;
              until turn = 2;
              outside1:= true;
              repeat until turn = 1;
            forever
          end
          P1 inside;
          turn:= 2; outside1:= true;
          P1 outside;
        forever

  "P2" repeat
          label enter
          begin
            repeat
              outside2:= false;
              repeat
                if outside1 then exit enter;
              until turn = 1;
              outside2:= true;
              repeat until turn = 2;
            forever
          end
          P2 inside;
          turn:= 1; outside2:= true;
          P2 outside;
        forever
  coend
end
```

CHAPTER 5: QUESTION TWO

The similarity between the two algorithms

CHAPTER 5: SOLUTION (2a)

Outline an informal argument (reasoning) which shows that If both processes are trying to enter their critical regions simultaneously, a decision will be made within a finite time as to which one should be permitted to do so.

- What they are asking is how do we guarantee that a deadlock does not occur?

- The variable turn is only changed at the end of a critical region; it can therefore be regarded as a constant when both processes are trying to enter their critical regions at the same time.

  If *turn* = 1, then process P1 can only cycle in the statement

  repeat
  if *outside2* then exit *enter;*
  until *turn* = 2;

  and process P2 can only cycle in the statement

  repeat until *turn* = 2;

- But the latter implies that *outside2* holds, so P1 will enter its region. A similar argument can be made when *turn* = *2.*

CHAPTER 5: SOLUTION (2b)

Outline an informal argument (reasoning) which shows that One process at most is inside its critical region at a time.

- What they are asking is how is mutual exclusion guaranteed?

- Notice that each process only changes its own variable *outside* and that
  outside1 implies P1 *outside &*
  outside2 implies P2 *outside*
- Since process P1 only enters its critical region when *outside2* holds (and vice versa for P2), mutual exclusion is guaranteed.

CHAPTER 5: SOLUTION (2c)

Outline an informal argument (reasoning) which shows that If a process is stopped outside its critical region, this cannot influence the progress of the other process.

- What they are asking is how do we guarantee the processes remain atomic?
- If P1 is stopped outside its critical region, we have

  outside1
- This will immediately permit process P2 to enter its critical region independent of the value of turn.

CHAPTER 5: QUESTION THREE

Consider the following program:

```
const int n = 50;
int tally;
void total()
{
    int count;
    for (count = 1; count<= n; count++) {
        tally++;
    }
}
 void main()
{
    tally = 0;
    parbegin (total (), total ());
    write (tally);
}
```

CHAPTER 5: QUESTION (3a)

Determine the proper lower bound and upper bound on the final value of the shared variable *tally* output by this concurrent program. Assume processes can execute at any relative speed, and a value can only be incremented after it has been loaded into a register by a separate machine instruction.

- On casual inspection, it appears that tally will fall in the range $50 \leq \text{tally} \leq 100$ since from 0 to 50 increments could go unrecorded due to the lack of mutual exclusion. The basic argument contends that by running these two processes concurrently we should not be able to derive a result lower than the result produced by executing just one of these processes sequentially.
- But consider the following interleaved sequence of the load, increment, and store operations in the table below

CHAPTER 5: QUESTION (3a)

		Value if tally Register A	Value of tally in Register B	Value Tally	n for A	n for B
1	Process A loads the value of tally, increments tally, but then loses the processor (it has incremented its register to 1, but has not yet stored this value.	1	_	0	1	0
2	Process B loads the value of tally (still zero) and performs forty-nine complete increment operations, losing the processor after it has stored the value 49 into the shared variable tally.	1	49	49	1	49
3	Process A regains control long enough to perform its first store operation (replacing the previous tally value of 49 with 1) but is then immediately forced to relinquish the processor.	1	49	1	1	49
4	Process B resumes long enough to load 1 (the current value of tally) into its register, but then it too is forced to give up the processor.	1	1	1	1	49
5	Process A is rescheduled, but this time it is not interrupted and runs to completion, performing its remaining 49 load, increment, and store operations, which results in setting the value of tally to 50.	50	1	50	50	49
6	Process B is reactivated with only one increment and store operation to perform before it terminates. It increments its register value to 2 and stores this value as the final value of the shared variable.	50	2	2	50	50

CHAPTER 5: QUESTION (3a)

- Some thought will reveal that a value lower than 2 cannot occur. Thus, the proper range of final values is $2 \leq \text{tally} \leq 100$.

CHAPTER 5: QUESTION (3b)

Suppose that an arbitrary number of these processes are permitted to execute in parallel under the assumptions of part (a). What effect will this modification have on the range of final values of tally?

- For the generalized case of N processes, the range of final values is $2 \leq \text{tally} \leq (N \times 50)$, since it is possible for all other processes to be initially scheduled and run to completion in step (5) before Process B would finally destroy their work by finishing last.

QUESTIONS?

COS122QUERIES@CS.UP.AC.ZA

Faculty of Engineering, Built Environment and Information Technology

Fakulteit Ingenieurswese, Bou-omgewing en Inligtingtegnologie / Lefapha la Boetšenere, Tikologo ya Kago le Theknolotši ya Tshedimošo

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Make today matter

www.up.ac.za