SOLUTIONS TO THE SECOND 3360/6310 QUIZ

Jehan-François Pâris
Spring 2016

First question

- In some implementations of monitors, the *signal* operation is replaced by a _____ operation.

First question

- In some implementations of monitors, the *signal* operation is replaced by a **notify** operation.

First question

- In Java, a monitor procedure must start with the two keywords **public** and _____

First question

- In Java, a monitor procedure must start with the two keywords **public** and **synchronized**

First question

- You should use *streams* rather than *datagrams* when you have to transmit _____ amounts of data.

First question

- You should use *streams* rather than *datagrams* when you have to transmit **large/huge** amounts of data.

First question

- The main disadvantage of the *round robin scheduling policy* is that it generates too many _____ at heavy loads.

First question

- The main disadvantage of the *round robin scheduling policy* is that it generates too many **context switches** at heavy loads.

First question

- The main disadvantage of *atomic transactions* is that they are _____

First question

- The main disadvantage of *atomic transactions* is that they are **very costly to implement**

First question

- A *signal* operation has *no effect* if there are no other monitor procedures

First question

- A *signal* operation has *no effect* if there are no other monitor procedures **waiting on the signaled condition**
- A *signal* operation has *no effect* if there are no other monitor procedures **waiting for that signal**

First question

- Unlike *virtual circuits*, *streams* do not preserve _____.

First question

- Unlike *virtual circuits*, *streams* do not preserve **message boundaries**

.

First question

- Priority-based scheduling algorithms typically assign _____ priorities to I/O-bound processes than to CPU-bound processes.

First question

- Priority-based scheduling algorithms typically assign **higher** priorities to I/O-bound processes than to CPU-bound processes.

First question

- _____ guarantee that each request will either be fully executed or have no effect.

First question

- **Atomic transactions** guarantee that each request will either be fully executed or have no effect.
- **All-nothings semantics** guarantee that each request will either be fully executed or have no effect.
- **Transactions semantics** guarantee that each request will either be fully executed or have no effect.

Second question

- Consider the function:
  - **double_decrement(int *pa, int *pb){**
    (*pa) -= 1; (*pb) -= 1;
    } //double_decrement
- and assume the following calling sequence:
  - **int alpha = 3;**
    double_decrement(&alpha, &alpha)

Second question

- What will be the value of **alpha *after the call*** assuming that the call was:
  - A ***regular procedure call?***
    - _____
  - A ***remote procedure call?***
    - _____

Second question

- What will be the value of **alpha *after the call*** assuming that the call was:
  - A ***regular procedure call?***
    - **3 – 1 – 1 = 1**
      - **alpha is decremented twice**
  - A ***remote procedure call?***
    - **3 – 1 = 2**
      - **alpha get assigned twice 3 -1 = 2**

Alternate second question

- Consider the function:
  - **double_decrement(int *pa, int *pb){**
    (*pa) -= 1; (*pb) -= 1;
    } //double_decrement
- and assume the following calling sequence:
  - **int alpha = 5; // 5 not 3**
    double_decrement(&alpha, &alpha)

Alternate second question

- What will be the value of **alpha *after the call*** assuming that the call was:
  - A ***regular procedure call?***
    - _____
  - A ***remote procedure call?***
    - _____

Alternate second question

- What will be the value of **alpha *after the call*** assuming that the call was:
  - A ***regular procedure call?***
    - **5 – 1 – 1 = 3**
      - **alpha is decremented twice**
  - A ***remote procedure call?***
    - **5 – 1 = 4**
      - **alpha get assigned twice 5 -1 = 4**

Third question

- Explain why ***lottery scheduling*** is inherently ***starvation free***?

Third question

- Explain why ***lottery scheduling*** is inherently ***starvation free***?
  - New tickets will not be issued until all processes that have thickets have used theirs

Third question

- What should a process do when it receives a ***retransmission*** of a message it has ***previously received*** and ***acknowledged*** ?

Third question

- What should a process do when it receives a ***retransmission*** of a message it has ***previously received*** and ***acknowledged*** ?
  - **It should acknowledge the retransmission (because otherwise the message will be resent a second time).**

Third question

- What is the difference between a ***blocking receive*** and a ***non-blocking receive***?

Third question

- What is the difference between a ***blocking receive*** and a ***non-blocking receive***?
  - **A blocking receive waits until the process receives a message while a non-blocking receive never waits.**

Third question

- When should we worry about ***little-endians*** and ***big-endians***?

Third question

- When should we worry about ***little-endians*** and ***big-endians***?
  - **We should worry about the byte ordering inside words each time we exchange data between two different machines.**

Third question

- What is the main disadvantage of ***spin locks***?

Third question

- What is the main disadvantage of ***spin locks***?
  - **Spin locks**
    - **Waste processor cycles**
    - **Generate additional context switches unless the waiting thread runs on a different core than the thread inside the critical section.**

Third question

- Why do most experienced programmers put their ***signal operations*** at the end of their ***monitor procedures***?

Third question

- Why do most experienced programmers put their ***signal operations*** at the end of their ***monitor procedures***?
  - **Because a thread performing a signal will lose its control of the monitor if a tread waiting on the condition catches the signal.**

Third question

- Explain why ***non-preemptive*** schedulers are ***bad***.

Third question

- Explain why ***non-preemptive*** schedulers are ***bad***.
  - **Non-preemptive scheduling policies do not prevent CPU-bound processes from monopolizing the CPU.**

Third question

- What is the difference between a ***blocking send*** and a ***non-blocking send***?

Third question

- What is the difference between a ***blocking send*** and a ***non-blocking send***?
  - **A blocking send waits until the message it sends had been delivered while a non-blocking send returns as soon as the message has been accepted for delivery.**

Third question

- When should we worry about the ***byte ordering*** inside computer words?

Third question

- When should we worry about the ***byte ordering*** inside computer words?
  - **We should worry about the byte ordering inside words each time we exchange data between two different machines.**

Third question

- When should we use ***spin locks*** instead of semaphore ***calls***?
  - **Spin locks are better for short waits whenever the waiting process and the process that holds the lock are on different core/on a multi-core computer**

Fourth question

- Consider the following system V scheduler:

#ts_q…	ts_tq…	ts_slp…	ts_max…	ts_lw…	LEVEL
800	U	1	4000	1	# 0
400	V	2	2000	2	# 1
200	1	W	1000	Y	# 2
100	2	X	500	Z	# 3

- Give correct values for the missing parameters:

Fourth question

- Consider the following system V scheduler:

#ts_q…	ts_tq…	ts_slp…	ts_max…	ts_lw…	LEVEL
800	U	1	4000	1	# 0
400	V	2	2000	2	# 1
200	1	W	1000	Y	# 2
100	2	X	500	Z	# 3

- Give correct values for the missing parameters:

U = V = 0 W = X = 3 Y = Z = 3

Fifth question

- An interstate bus that can carry 80 passengers has a single door that lets passengers get in or out one by one at scheduled stops. Add semaphores to the following two functions to ensure that
  - The bus will never have more than 80 passengers and
  - Passengers will not collide with each other when getting through the bus door.

Fifth question

- ```
  semaphore notfull =   80   ;
  semaphore door    =   1    ;
  ```
- ```
  get_in(){
      P(&freeseats); // in that order
      P(&door);      // in that order
      go_through_door();
      V(&door);

      ____________________

  } //get in
  ```

Fifth question

- ```
  get_out(){
      _____________________
      P(&door);
      go_through_door();
      V(&door);        // in any order
      V(&freeseats); // in any order
  } //get_out
  ```
  ```

Alternate fifth question

- An interstate bus that can carry 30 passengers has a single door that lets passengers get in or out one by one at scheduled stops. Add semaphores to the following two functions to ensure that
  - The bus will never have more than 30 passengers and
  - Passengers will not collide with each other when getting through the bus door.

Alternate fifth question

- semaphore notfull = 80 ;
  semaphore door = 1 ;
- get_in(){
  P(&freeseats); // in that order
  P(&door); // in that order
  go_through_door();
  V(&door);

  }_//get_in

Alternate fifth question

```
get_out(){
    _____________
    P(&door);
    go_through_door();
    V(&door);        // in any order
    V(&freeseats); // in any order
} //get_out
```