**Spring 2019 COMP 3511 Homework Assignment #3**

**Handout Date: March 30, 2019 Due Date: April 12, 2019**

Name: \_HE Jianle\_\_\_\_\_\_ ID: \_\_20536134\_\_\_\_\_\_E-Mail: \_\_jhebg@ust.hk\_

**Please read the following instructions carefully before answering the questions:**

* You should finish the homework assignment **individually**.
* There are total of **4** questions.
* When you write your answers, please try to be precise and concise.
* Fill in your name, student ID, email and Section number at the top of each page.
* Please fill in your answers in the space provided.
* **Homework Collection:** the homework is submitted to **assignment #3** on **CASS**

1. [20 points] Multiple choices
   1. The first readers-writers problem \_\_\_\_.

A) requires that, once a writer is ready, that writer performs its write as soon as possible.

B) is not used to test synchronization primitives.

C) requires that no reader will be kept waiting unless a writer has already obtained permission to use the shared database.

D) requires that no reader will be kept waiting unless a reader has already obtained permission to use the shared database.

**Answer**: \_C\_

* 1. Which of the following conditions must be satisfied to solve the critical section problem?

①Aging ②Mutual Exclusion ③Deadlock ④Progress ⑤Bounded Waiting

A) ①②③⑤

B) ②③④⑤

C) ②④⑤

D) ③④⑤

**Answer**: \_C\_

* 1. Assume an adaptive mutex is used for accessing shared data on a Solaris system with multiprocessing capabilities. Which of the following statements is not true?

A) A waiting thread may spin while waiting for the lock to become available.

B) A waiting thread may sleep while waiting for the lock to become available.

C) The adaptive mutex is only used to protect short segments of code.

D) Condition variables and semaphores are never used in place of an adaptive mutex.

**Answer**: \_D\_

* 1. \_\_\_\_\_\_\_\_\_\_\_\_ occurs when a higher-priority process needs to access a data structure that is currently being accessed by a lower-priority process.

A) Deadlock

B) Priority inversion

C) A race condition

D) A critical section

**Answer**: \_B\_

* 1. A deadlocked state occurs whenever \_\_\_\_.

A) a process is waiting for I/O to a device that does not exist

B) the system has no available free resources

C) every process in a set is waiting for an event that can only be caused by another process in the set

D) a process is unable to release its request for a resource after use

**Answer**: \_C\_

* 1. Which of the following condition is required for deadlock to be possible?

A) Mutual exclusion

B) A process may hold allocated resources while awaiting assignment of other resources.

C) No resource can be forcibly removed from a process holding it.

D) All of the mentioned.

**Answer**: \_D\_

* 1. Suppose that there are three processes and ten resources of the same type. The current resource allocation and the maximum need of each process is given below, which of the following correctly characterizes this state?

Process Maximum Needs Current Allocation

P0 10 4

P1 3 1

P2 6 4

A) It is not safe.

B) It is safe.

C) The state cannot be determined.

D) It is an impossible state.

**Answer**: \_A\_

* 1. Absolute code can be generated for \_\_\_\_.

A) compile-time binding

B) load-time binding

C) execution-time binding

D) interrupt binding

**Answer**: \_A\_

* 1. Which of the following is true of compaction?

A) It can be done at assembly, load, or execution time.

B) It is used to solve the problem of internal fragmentation.

C) It cannot shuffle memory contents.

D) It is possible only if relocation is dynamic and done at execution time.

**Answer**: \_D\_

* 1. \_\_\_\_\_ is the dynamic storage-allocation algorithm which results in the largest leftover hole in memory.

A) First fit

B) Best fit

C) Worst fit

D) None of the above

**Answer**: \_C\_

1. [30 points] Synchronization
   1. Some semaphore implementations provide a function getValue() that returns the current value of a semaphore. This function may, for instance, be invoked prior to calling wait() so that a process will only call wait() if the value of the semaphore is > 0, thereby preventing blocking while waiting for the semaphore. For example:

|  |
| --- |
| if (getValue(&sem) > 0)  wait(&sem); |

What is the problem in this approach? (5 points)

Lack of mutually exclusive condition. The wait() be should waiting for getValue(&sem) > 0, so that whenever the wait() function be invoked, the process be exit the wait() directly. On the other hand, whenever the getValue(&sem) <= 0, the process should wait but never wait.

* 1. Briefly describe what a reader-writer lock is for and why it can be more efficient than semaphores in some cases. (5 points)

reader-writer lock can allow multiple reader access at the same time. However, only one writer can enter the critical section (no other reader or writer). Therefore, if there is no process going to update the value of shared data, everybody can read the shared data at the same time, in other word improve the efficiency.

* 1. Considering the first reader-writer solution below, please explain which semaphores that readers and writers are waiting on when there is a writer in inside the Critical Section updating shared data. (5 points)

|  |  |
| --- | --- |
| **Writer**  do {  wait(rw\_mutex);  ...  /\* writing is performed \*/  ...  signal(rw\_mutex);  } while (true); | **Reader**  do {  wait(mutex);  read\_count++;  if (read\_count == 1)  wait(rw\_mutex);  signal(mutex)  ...  /\* reading is performed \*/  ...  wait(mutex);  read\_count--;  if (read\_count == 0)  signal(rw\_mutex);  signal(mutex);  } while (true); |

Only one writer can enter the Critical Section every time, so except the writer inside the Critical Section, all other writers are waiting on the rw\_mutex. Writers can enter Critical Section only if read\_count == 0, that means the first reader is also waiting on the rw\_mutex. All other readers are waiting for the first reader on mutex.

* 1. Given a condition variable x, Consider the following implementation of x.signal() using semaphores. Please explain whether this a Hoare monitor or Mesa monitor, and why? (5 points)

|  |
| --- |
| if (x\_count > 0) {  next\_count++;  signal(x\_sem);  wait(next);  next\_count--;  } |

Hoare monitor. let P be current process and Q be waking up process. P give up monitor and wait on the next queue while P wake Q up by signal(x\_sem). Therefore is signal and wait, a Hoare monitor.

* 1. You are asked to implement a different reader-writer solution. There are two classes of processes accessing shared data, *readers* and *writers*. Readers never modify data, thus multiple readers can access the shared data simultaneously. Writers modify shared data, so at most one writer can access data (no other writers or readers). This solution gives priority to writers in the following manner: when a reader tries to access shared data, if there is a writer accessing the data or if there are any writer(s) waiting to access shared data, the reader must wait. In another word, readers must wait for all writer(s) if any to update shared data -- a reader can access shared data only when there is no writer either accessing or waiting.

Variables:

|  |
| --- |
| State variables (protected by a lock called “lock”  condition okToRead = NIL; /\* readers waiting queue \*/  condition okToWrite = NIL; /\* writers waiting queue \*/  int R\_count = 0; /\* number of readers accessing data \*/  int W\_count = 0; /\* number of writer accessing data \*/  int WR\_count = 0; /\* number of readers waiting \*/  int WW\_count = 0; /\* number of writers waiting\*/ |

The writer code is given below. Please design the Reader’s code. (10 points)

|  |
| --- |
| Writer() {  // Writer tries to enter  lock.acquire;  while ((R\_count + W\_count) > 0) {// Is it safe to write?  WW\_count++; // Update the counter of waiting writers  okToWrite.wait(&lock); // Waiting on condition variable,  atomically release the lock, regain the lock later  WW\_count--; // No longer waiting  }  W\_count++; // Writer inside  lock.release();  // Perform actual read/write access  // Writer finishes update  lock.acquire();  W\_count--; // No longer active  if (WW\_count > 0){ // Give priority to writers  okToWrite.signal(); // Wake up one writer  } else if (WR\_count > 0) {// Otherwise, wake up readers  okToRead.broadcast(); // Wake up all waiting readers  }  lock.relesae();  } |

**Answer**:

|  |
| --- |
| Reader() {  // Reader tries to enter  lock.acquire;  while (W\_count > 0 || WW\_count > 0) {// Is it safe to read?  WR\_count++; // Update the counter of waiting readers  okToRead.wait(&lock); // Reading on condition variable, atomically release the lock, regain the lock later  WR\_count--; // No longer waiting  }  R\_count++; // Reader inside  lock.release();  // Perform actual read access  // Reader finishes reading  lock.acquire();  R\_count--; // No longer active  if (R\_count == 0 && WW\_count > 0) {// Give priority to writers  okToWrite.signal(); // Wake up one writer  }  lock.release();  } |

1. [30 points] Deadlock
   1. What does a deadlock prevention mechanism do? Use an example to illustrate why this can lead to low resource utilization. (5 points)

There are 4 necessary factors for deadlock. Deadlock prevention is to modify any one the 4 factors, therefore no deadlock exists. In case of hold and wait, the process has to wait until all the requested resources are obtained. This may increase the waiting time.

* 1. Please briefly explain the two methods of deadlock recovery. (5 points)

Abort all the deadlocked processes: this simply kill all processes that stuck in deadlock.

Rollback: return to a previous state that is in safe state, and restart the process from the state.

* 1. Consider a system with three processes and twelve instances of one type of resource. The current resource allocation and the maximum need of each process is given below, please find a safe sequence. (5 points)

Process Maximum Needs Current Allocation

P0 10 5

P1 4 2

P2 9 2

P1 -> P0 -> P2

* 1. Consider the following snapshot of a system:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Allocation | Max | Available |
|  | A B C D | A B C D | A B C D |
| P0 | 0 0 1 1 | 1 0 2 2 | 1 2 1 1 |
| P1 | 3 0 4 0 | 5 6 6 2 |  |
| P2 | 1 0 2 0 | 6 4 3 1 |  |
| P3 | 1 1 0 0 | 1 2 2 1 |  |
| P4 | 0 4 1 1 | 2 4 3 3 |  |

Answer the following questions using the banker’s algorithm (15 points)

* + 1. Illustrate that the system is in a safe state by demonstrating an order in which the processes may complete. (5 points)

The processes will complete in the order: P0 – P3 – P4 – P1 – P2

* + 1. If a request from process P4 arrives for (1, 0, 0, 0), can the request be granted immediately? (5 points)

No, if P4 is granted, the available will be (0, 2, 1, 1), and deadlock will occur in all other processes.

* + 1. If a request from process P1 arrives for (0, 1, 0, 0), can the request be granted immediately? (5 points)

Yes, if P1 is granted, the available will be (1, 1, 1, 1), there is no any impact to other processes, no deadlock will be occur.

1. [20 points] Memory management
   1. Briefly describe internal and external fragmentation, and methods to mitigate the problems. (5 points)

external fragmentation: total memory space exists to satisfy a request, but it is not contiguous

internal fragmentation: memory allocated to a process may be larger than requested memory; this size difference is memory internal to a partition, but not being used

segmentation and paging are able to mitigate the problem.

* 1. Consider the following segment table:

|  |  |  |
| --- | --- | --- |
| Segment  0  1  2  3  4 | Base  0000010000000000  0010000000100000  0011000000000000  0100000000000010  1000000000001000 | Length  001011101110  011110011110  010100011110  010110001100  010100010110 |

Consider the following 16-bit logical addresses with 4-bit segment and 12-bit offset, what are the physical addresses of them? (15 points)

1. 0001001011110000 0010001100010000
2. 0000100011101110 addressing error
3. 0010010100010000 0011010100010000
4. 0011010010000000 0100010010000010
5. 0100000100010000 1000000100011000