TA responsible for HW3: Xingxing TANG (xtangav@connect.ust.hk)

Fall 2024 COMP 3511 Homework Assignment #3

Handout Date: October 30 2024, Due Date: November 13 2024

Name	**LI,Yuntong**
Student ID	**20944800**
ITSC email	ylino@connect.ust.hk

Please read the following instructions carefully before answering the questions:

- You must finish the homework assignment **individually**.
- When you write your answers, please try to be precise and concise.
- **Homework Submission:** Please submit your homework to **Homework #3** on **Canvas**.
- TA responsible for HW3: Xingxing TANG (xtangav@connect.ust.hk)

1. (30 points) Multiple Choices

Write your answers in the boxes below:

MC1	MC2	MC3	MC4	MC5	MC6	MC7	MC8	MC9	MC10
D	C	C	C	D	D	D	D	A	A

(1) Which of the following statements is TRUE about *Race Condition*?

A) Race Condition can cause potential data inconsistency

B) The outcome of the executions depends on the particular order

C) Even race condition exists, the outcome of the executions can be correct

D) All of the above

(2) Which of the following statements is TRUE about *Mutex Lock*?

A) Calling acquire() of a mutex lock can be interrupted

B) Calling release() of a mutex lock can be interrupted

C) Mutex lock is often implemented as a spinlock

D) All of the above

(3) Which of the following statements is FALSE about *Semaphore*?

A) Improper usage of semaphore can cause deadlock

B) Semaphore can be used as a mutex lock

C) Semaphore values cannot become negative

D) Semaphore has busy waiting and non-busy waiting versions

(4) Which of the following statements is TRUE about *Reader-Writer Problems*?

A) Multiple readers cannot read the shared data simultaneously

TA responsible for HW3: Xingxing TANG (xtangav@connect.ust.hk)

B) We should always prioritize readers in reader-writer problem

C) OS usually provides a reader-writer lock to mitigate the starvation problem

D) None of the above

(5) Which of the following statements is TRUE about *Deadlock*?

A) Deadlock prevention and avoidance guarantee that deadlock will never happen

B) Circular wait always leads to deadlock

C) The number of resource instances has no impact on whether the deadlock will happen

D) None of the above

(6) Which of the following statement is TRUE about the *Bankers' Algorithm*?

A) Bankers' algorithm does not need to know the maximum resource usage for each process

B) The Safety Algorithm is only invoked once at the beginning of the algorithm

C) After resource is allocated using Resource-Request Algorithm, we never restore the state of resource allocation

D) None of the above

(7) Which of the following statement is FALSE about the *Deadlock Detection Algorithm*?

A) The deadlock detection algorithm is identical to the safety algorithm

B) Using deadlock detection algorithm for each resource allocation request causes large overhead

C) The deadlock detection algorithm can tell which process cause the deadlock

D) All of the above

(8) Consider a segment table of one process:

Segment #	Segment length	Starting address	Permission	Status
0	100	6000	Read-only	In memory
1	150	5500	Read/Write	In memory
2	350	4000	Read/Write	In memory

When accessing the logical address at , the results after address translation is:

A) No such segment

B) Return address 4400

C) Invalid permission

D) Address out of range

(9) In a system with 32-bit address, given the logical address 0x0000F1BC (in hexadecimal) with a page size of 256 bytes, what is the page offset?

A) 0xBC

B) 0xF1

TA responsible for HW3: Xingxing TANG (xtangav@connect.ust.hk)

C) 0xC

D) 0xF100

(10) Which of the following statement is FALSE when comparing Segmentation and Paging Schemes?

A) The number of entries in a page table is usually smaller than the number of entries in a segmentation table

B) Paging scheme results in better memory utilization than segmentation scheme

C) Sharing a segment is easier that sharing a page, as each segment represents a logical entity in a program

D) Paging scheme does not suffer from external fragmentation

2. (20 points) Process Synchronization

This problem involves multiple readers and one writer accessing a shared resource. When the writer wants to write, readers must wait, ensuring the writer gets priority.

We will use a new API *pthread_cond_broadcast(&cond)* to wake up all waiting threads on the conditional variable *cond*.

Please fill in 10 blanks to correctly implement synchronization using mutex locks and condition variables.

```
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>

int read_count = 0;          // Number of active readers
int writer_waiting = 0;      // Is the writer waiting?

pthread_mutex_t read_count_mutex;  // Protects the read_count variable
pthread_mutex_t writer_mutex;     // Controls writer's priority and access
pthread_cond_t reader_cond;       // Condition variable for readers
pthread_cond_t writer_cond;       // Condition variable for the writer

void* reader(void* arg) {
  // Lock to modify read_count
  pthread_mutex_lock(&read_count_mutex);  // (1) Lock read_count_mutex to access read_count

  // Wait if the writer is waiting
  while (writer_waiting > 0) {  // (2) Check if the writer is waiting
    pthread_cond_wait(&reader_cond, &read_count_mutex);  // (3) Wait for the writer
  }
  read_count++;  // Increment reader count
```

```
    pthread_mutex_unlock(&read_count_mutex);  // Unlock read_count_mutex

    // Reading section
    printf("Reader %d is reading.\n", id);

    // Lock to decrement read_count
    pthread_mutex_lock(&read_count_mutex);  // Lock read_count_mutex
    read_count--;  // Decrement reader count

    // If no readers are active, signal the writer
    if (read_count == 0) {  // (4) Check if no readers are active
        pthread_cond_signal(&writer_cond);  // (5) Signal the writer
    }
    pthread_mutex_unlock(&read_count_mutex);  // Unlock read_count_mutex

    return NULL;
}

void* writer(void* arg) {
    // Lock to manage writer state and access
    pthread_mutex_lock(&writer_mutex);  // (6) Lock writer_mutex to access writer state
    writer_waiting = 1;  // Mark the writer as waiting

    // Wait until all readers are done
    pthread_mutex_lock(&read_count_mutex);  // Lock read_count_mutex
    while (read_count > 0) {  // (7) Check if any readers are active
        pthread_cond_wait(&writer_cond, &read_count_mutex);  // (8) Wait for readers to finish
    }
    pthread_mutex_unlock(&read_count_mutex);  // Unlock read_count_mutex

    // Writing section
    printf("Writer %d is writing.\n", id);
    writer_waiting = 0;  // Mark the writer as not waiting

    // Signal waiting readers to proceed
    pthread_cond_broadcast(&reader_cond);  // (9) Broadcast to readers
    pthread_mutex_unlock(&writer_mutex);  // (10) Unlock writer_mutex

    return NULL;
```

TA responsible for HW3: Xingxing TANG (xtangav@connect.ust.hk)

}

3. (30 points) Deadlocks

Consider the following snapshot of a system:

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

(1) (5 points) What is the content of the Need matrix?

	A	B	C	D
P0	2	2	3	2
P1	2	0	1	1
P2	1	3	1	3
P3	0	1	1	2
P4	2	0	1	0

(2) (10 points) Is the system in a safe state? Why?

Yes, the system is in a safe state.

Work = Available resources = [2, 3, 2, 1]

- P0: Need = [2, 2, 3, 2] > Work
- P1: Need = [2, 0, 1, 1] ≤ Work

Work = [2, 3, 2, 1] + [4, 1, 2, 1] = [6, 4, 4, 2]

- P2: Need = [1, 3, 1, 3] > Work
- P3: Need = [0, 1, 1, 2] ≤ Work

Work = [6, 4, 4, 2] + [1, 2, 1, 2] = [7, 6, 5, 4]

- P4: Need = [2, 0, 1, 0] ≤ Work

Work = [7, 6, 5, 4] + [1, 4, 5, 1] = [8, 10, 10, 5]

- P2: Need = [1, 3, 1, 3] ≤ Work

Work = [8, 10, 10, 5] + [1, 1, 0, 3] = [9, 11, 10, 8]

- P0: Need = [2, 2, 3, 2] ≤ Work

Work = [9, 11, 10, 8] + [2, 0, 0, 1] = [11, 11, 10, 9]

Since we have found a sequence in which all processes can finish (P1 → P3 → P4 → P2 → P0), the system is in a safe state.

TA responsible for HW3: Xingxing TANG (xtangav@connect.ust.hk)

(3) (5 points) If a request from process P1 arrives for (1, 1, 0, 2), can the request be granted? Why?

No, because (1, 1, 0, 2) > Need[1]=(2, 0, 1, 1) at resource B
So this request is not allowed, and process P1 should be blocked.

(4) (10 points) If a request from process P0 arrives for (1, 1, 2, 1), can the request be granted? Why?

The request cannot be granted.

The request [1, 1, 2, 1] <= Need [2, 2, 3, 2], and [1, 1, 2, 1] <= Available [2, 3, 2, 1]

Next,
- the Updated Allocation Matrix: [[3, 1, 2, 2] [4, 1, 2, 1] [1, 1, 0, 3] [1, 2, 1, 2] [1, 4, 5, 1]]
- the Updated Need Matrix: [[1, 1, 1, 1] [2, 0, 1, 1] [1, 3, 1, 3] [0, 1, 1, 2] [2, 0, 1, 0]]

Since Available Resources [1, 2, 0, 0] is smaller than any Need from each process, the request from process P0 for (1, 1, 2, 1) cannot be granted.

TA responsible for HW3: Xingxing TANG (xtangav@connect.ust.hk)

4. (20 points) Memory Management

Consider the segment table shown in Table A. Translate each of the virtual addresses in Table B into physical addresses. Indicate errors (out of range, no such segment) if an address cannot be translated.

Table A

Segment number	Starting address	Segment length
0	260	90
1	1466	160
2	2656	130
3	146	50
4	2064	370

Table B

Segment number	Offset
0	10
1	180
2	100
3	80
4	50
5	32

Answer:

Segment number	Offset	Physical address
0	10	270
1	180	Out of range
2	100	2756
3	80	Out of range
4	50	2114
5	32	No such segment

TA responsible for HW3: Xingxing TANG (xtangav@connect.ust.hk)

5. (12 points) Paging

Consider a virtual memory system providing 256 pages for each user program; the size of each page is 4KB. The size of main memory is 256KB. Consider one user program occupied 4 pages, and the page table of this program is shown as below:

Logical page number	Physical block number
0	9
1	6
2	7
3	3

Assume there are two requests on logical address (in hexadecimal number) 010AF, 100AF.

(1) (4 points) Please describe how many bits the virtual page number and the page offset contain, respectively.

Virtual page number: 8 bits (since 256 pages = 2^8)

Page offset: 12 bits (since 4KB = 2^12 bytes)

(2) (8 points) Please illustrate how the virtual memory system will deal with these requests. (Please indicate if there will be a page fault. If there is no page fault, please give the corresponding physical address.)

(010AF):

Binary: 0000 0001 0000 1010 1111

Virtual Page Number: 00000001

1 exists in Page Table, maps to Physical Block Number 6.

Page Offset: 0000 1010 1111 (0x0AF)

The physical address will be 0x060AF.

(100AF): Page fault occurs, as page number 16 is not mapped.

TA responsible for HW3: Xingxing TANG (xtangav@connect.ust.hk)