Synchronization

1. [Based on [1] pg. 144, #8] Consider a machine instruction, SWAP(X, Y), which swaps the contents of memory locations X and Y in a single indivisible operation. In other words, it is defined as follows:

   ```c
   SWAP(int *X, int *Y)
   ```

   1: int temp
   2: temp = *X
   3: *X = *Y
   4: *Y = temp

   Use SWAP to implement the P(s) and V(s) operations of binary semaphores.

2. In the synchronization problems we’ve worked on before, multiple processes or threads are all modifying shared data. However, it is possible to get problems even if only one thread is modifying the data while others read it.

   Describe a faulty scenario that can occur if the `linkedlist_printlist` function is not protected by locks, even though all other functions on Linked Lists are protected by a shared lock. Recall that we have the following functions on Linked Lists:

   ```
   • Linked_List *linkedlist_create()
   • void linkedlist_destroy(Linked_List *l)
   • void linkedlist_insert(Linked_List *l, void *item, int key)
   • void linkedlist_prepend(Linked_List *l, void *item)
   • void *linkedlist_removehead(Linked_List *l, int *key)
   • void linkedlist_printlist(Linked_List *l, int which)
   ```

   Also recall that a scenario is faulty only if it is not equivalent to some sequential order of calls to Linked List functions.

Scheduling

1. [From [1], pg. 173, #3 & #4] Consider the following set of five processes where arrival is the time the process becomes ready, t is the total service time, and e is the external priority.

<table>
<thead>
<tr>
<th>process</th>
<th>arrival</th>
<th>t</th>
<th>e</th>
</tr>
</thead>
<tbody>
<tr>
<td>p0</td>
<td>0</td>
<td>80</td>
<td>9</td>
</tr>
<tr>
<td>p1</td>
<td>15</td>
<td>25</td>
<td>10</td>
</tr>
<tr>
<td>p2</td>
<td>15</td>
<td>15</td>
<td>9</td>
</tr>
<tr>
<td>p3</td>
<td>85</td>
<td>25</td>
<td>10</td>
</tr>
<tr>
<td>p4</td>
<td>90</td>
<td>10</td>
<td>11</td>
</tr>
</tbody>
</table>

   Assume that execution starts immediately at time 0 and there is no context switch overhead.
(a) For the following scheduling disciplines, draw a time diagram showing when each of the five processes executes. In the case of a tie, assume that the process with the lower process number goes first.

i. FIFO
ii. SJF
iii. SRT
iv. RR (quantum = 1)
v. ML (with FIFO at each priority queue)

(b) For each of these scheduling disciplines, compute the average turnaround time for the five processes.

Deadlock

1. [From [1], pg. 200, #1] Consider the dining philosophers problem. Assume there are three philosophers, p1, p2, and p3 using three forks f1, f2, and f3. The philosophers execute the following code:

```c
p1: while(1) {
    P(f1);
    P(f3);
    eat;
    V(f3);
    V(f1);
}
p2: while(1) {
    P(f1);
    P(f2);
    eat;
    V(f2);
    V(f1);
}
p3: while(1) {
    P(f3);
    P(f2);
    eat;
    V(f2);
    V(f3);
}
```

(a) Is deadlock possible in this system?
(b) Would deadlock be possible if we reversed the order of the P operations in p1, p2, or p3?
(c) Would deadlock be possible if we reversed the order of the V operations in p1, p2, or p3?

Memory Management

1. For each of the following memory management approaches, determine whether memory managed using the approach can have internal fragmentation, external fragmentation, neither, or both.

   (a) Fixed partitioning.
   (b) Variable partitioning.
   (c) Paged memory.

2. [From [1], page 229, #9] Consider a system with 640K of main memory. The following blocks are to be requested and released:

<table>
<thead>
<tr>
<th>block</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>size(KB)</td>
<td>200</td>
<td>200</td>
<td>150</td>
<td>80</td>
<td>80</td>
<td>240</td>
</tr>
</tbody>
</table>

Obviously, not all blocks can be accommodated at the same time. We say an allocation failure occurs when there is no hole large enough to satisfy a given request. Devise a sequence of requests and releases that results in an allocation failure under:
(a) the first fit policy, but not the best fit policy
(b) the best fit policy, but not the first fit policy

For each of the two situations, draw a diagram that shows what parts of memory are allocated at the time of failure. (Hint: don’t waste time trying to find a systematic approach to this problem – simply experiment with different sequences.)

Recall that in best fit, we allocate a block in the smallest hole that will accommodate it. In first fit, we search from the beginning of memory to find the first hole that will accommodate the request.

**Virtual Memory**

1. In terms of the address map function that must be implemented in hardware, what is the difference between standard paging and demand paging?

2. Implementing LRU. Consider the following two implementations of the Least-Recently Used (LRU) page-replacement algorithm:

   • **Counters.** For each page-table entry, we keep a 64-bit time-of-use register. We also keep a logical clock counter for the CPU. On every memory access, the CPU increments the clock counter and copies it to the time-of-use register for the page being accessed.

   • **Stack.** We maintain a stack of references to page table entries. Whenever a page is referenced, its entry is removed from the stack and put on the top of the stack. The stack is usually implemented as a doubly-linked list (with head and tail pointers) since we need to be able to remove entries from the middle.

(a) How is the LRU victim chosen if we use counters? Assume that the clock counter does not overflow.

(b) How is the LRU victim chosen if we use a stack? Assume that the stack fits entirely in memory.

(c) For the counter approach, which of the following, if any, is likely to be faster than the other in the worst case. Explain.
   i. Updating the data structures when a page is referenced
   ii. Finding a victim

(d) For the stack approach, which of the following, if any, is likely to be faster than the other in the worst case. Explain.
   i. Updating the data structures when a page is referenced
   ii. Finding a victim

**References**