7.8

Problem

The Linux kernel has a policy that a process cannot hold a spinlock while attempting to acquire a semaphore. Explain why this policy is in place.

Answer

Since obtaining a semaphore can cause a process to enter a sleep state while it awaits the semaphore's availability, it's crucial to limit the duration of holding spinlocks. Sleeping processes might retain spinlocks for extended periods, which is undesirable as spinlocks are intended for short-term usage.

8.20

Problem

In a real computer system, neither the resources available nor the demands of processes for resources are consistent over long periods (months). Resources break or are replaced, new processes come and go, and new resources are bought and added to the system. – If deadlock is controlled by the banker’s algorithm, which of the following changes can be made safely (without introducing the possibility of deadlock), and under what circumstances? – (a) Increase Available (new resources added). – (b) Decrease Available (resource permanently removed from system). – (c) Increase Max for one process (the process needs or wants more resources than allowed). – (d) Decrease Max for one process (the process decides that it does not need that many resources). – (e) Increase the number of processes. – (f) Decrease the number of processes.

Answer

(a) Increasing the available resources can help reduce the likelihood of deadlock. (b) Decreasing the available resources can increase the likelihood of deadlock.

(c) Increasing the maximum resources for a process can lead to deadlock. (d) Decreasing the maximum resources for a process can help reduce the likelihood of deadlock.

(e) Increasing the number of processes can increase the likelihood of deadlock. (f) Decreasing the number of processes can help reduce the likelihood of deadlock.

8.27

Problem

Consider the following snapshot of a system:

P	Allocation	Max

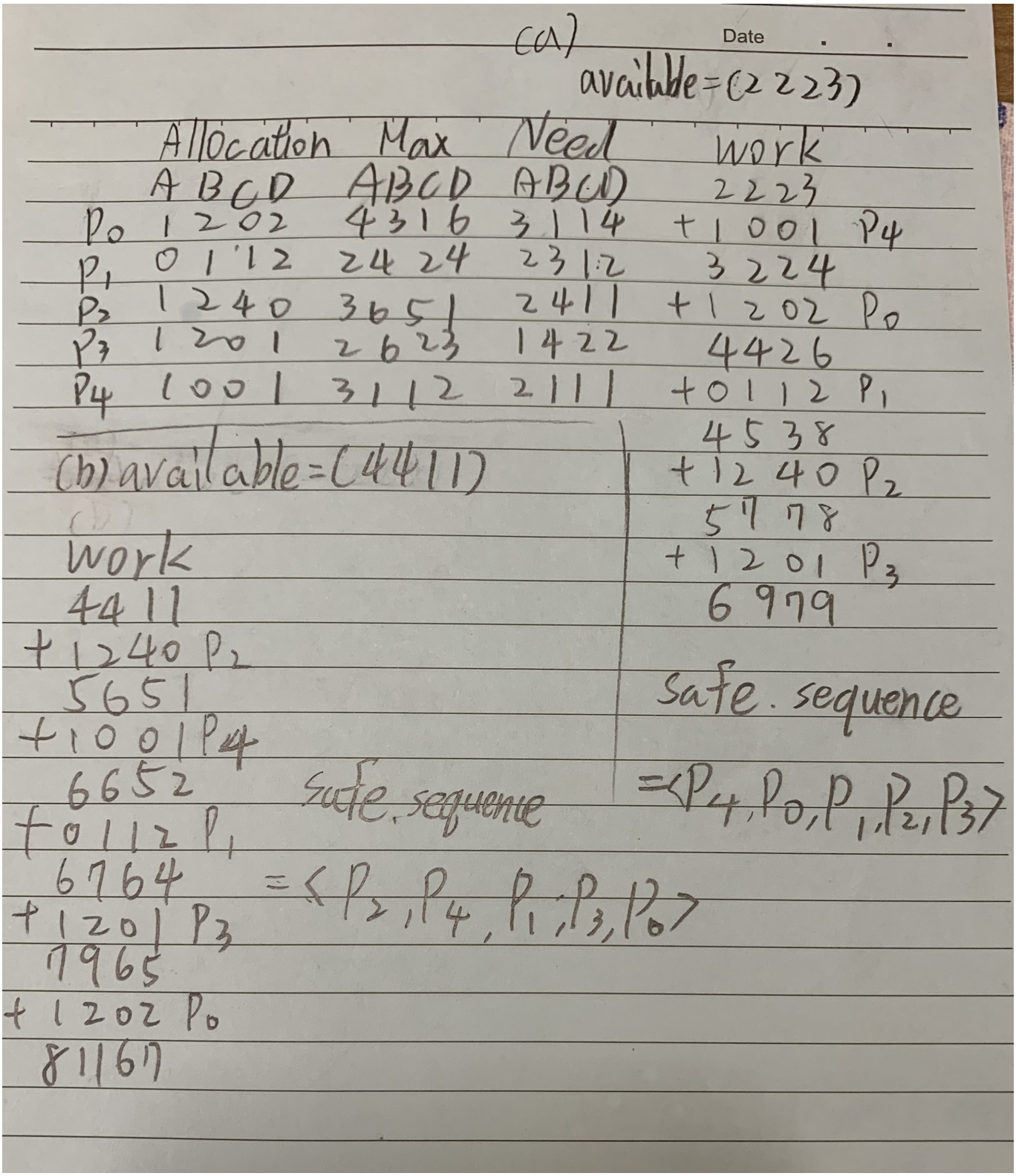
P	Allocation	Max
--	A B C D	A B C D
P0	1 2 0 2	4 3 1 6
P1	0 1 1 2	2 4 2 4
P2	1 2 4 0	3 6 5 1
P3	1 2 0 1	2 6 2 3
P4	1 0 0 1	3 1 1 2

Use the banker’s algorithm, determine whether or not each of the following states is unsafe. If the state is safe, illustrate the order in which the processes may complete.Otherwise, illustrate why the state is unsafe.

(a) Available = (2, 2, 2, 3) (b) Available = (4, 4, 1, 1)

Answer

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



(a) Date

available = (2223)

	Allocation	Max	Need	work
	ABCD	ABCD	ABCD	2223
P_0	1202	4316	3114	+1001 P_4
P_1	0112	2424	2312	3224
P_2	1240	3651	2411	+1202 P_0
P_3	1201	2623	1422	4426
P_4	1001	3112	2111	+0112 P_1
				4538
				+1240 P_2
				5778
				+1201 P_3
				6979

safe. sequence

$= \langle P_4, P_0, P_1, P_2, P_3 \rangle$

(b) available = (4411)

work

4411

+1240 P_2

5651

+1001 P_4

6652

+0112 P_1

6764

+1201 P_3

7965

+1202 P_0

8167

safe. sequence

$= \langle P_2, P_4, P_1, P_3, P_0 \rangle$

8.30

Problem

A single-lane bridge connects the two Vermont villages of North Tunbridge and South Tunbridge. Farmers in the two villages use this bridge to deliver their produce to the neighbor town.

- The bridge can become deadlocked if a northbound and a southbound farmer get on the bridge at the same time. (Vermont farmers are stubborn and are unable to back up.)

- Using semaphores and/or mutex locks, design an algorithm in pseudocode that prevents deadlock.
- Initially, do not be concerned about starvation (the situation in which northbound farmers prevent southbound farmers from using the bridge, or vice versa).

Answer

Assume that the bridge is a shared resource that can only be used by one farmer at a time. The following pseudocode demonstrates how to prevent deadlock by using semaphores and mutex locks.

```
mutex_t mutex; // mutex lock to protect the bridge resource
void cross_bridge() {
    // cross the bridge
}
void northbound_farmer() {
    while (true) {
        lock(mutex); // acquire the bridge resource
        cross_bridge(); // cross the bridge from North Tunbridge to South
Tunbridge
        unlock(mutex); // release the bridge resource
    }
}

void southbound_farmer() {
    while (true) {
        lock(mutex); // acquire the bridge resource
        cross_bridge(); // cross the bridge from South Tunbridge to North
Tunbridge
        unlock(mutex); // release the bridge resource
    }
}

void main() {
    initialize(mutex); // initialize the mutex lock
    create_thread(northbound_farmer); // create a thread for northbound
farmers
    create_thread(southbound_farmer); // create a thread for southbound
farmers
    join_thread(northbound_farmer); // join the northbound farmer thread
    join_thread(southbound_farmer); // join the southbound farmer thread
}
```

9.15

Problem

Compare the memory organization schemes of contiguous memory allocation and paging with respect to the following issues:

- (a) external fragmentation

- (b) internal fragmentation
- (c) ability to share code across processes

Answer

(a) External fragmentation: Contiguous memory allocation with fixed partitions and paging does not suffer from external fragmentation. However, contiguous memory allocation with variable partitions can experience external fragmentation when processes are loaded and removed from memory, leading to inefficient memory utilization.

(b) Internal fragmentation: contiguous memory allocation with fixed partitions and paging does suffer from internal fragmentation. In contiguous memory allocation with fixed partitions, processes may not fully utilize the allocated memory block, leading to internal fragmentation. In paging, the last page of a process may not be fully utilized, resulting in internal fragmentation. Yet, contiguous memory allocation with variable partitions will not have internal fragmentation.

(c) Ability to share code across processes: Either two version of contiguous memory allocation can not share code across processes. However, paging can share code across processes. This is accomplished by mapping different logical addresses to the same physical address, allowing multiple processes to access the same code segment in memory.

9.24

Problem

Consider a computer system with a 32-bit logical address and 8-KB page size. The system supports up to 1 GB of physical memory. How many entries are there in each of the following?

- (a) A conventional, single-level page table
- (b) An inverted page table

Answer

Both a conventional, single-level page table and an inverted page table in this scenario would have 128 entries each. Given:

Logical address size: 32 bits(2 ^ 32) Page size: 8 KB (2 ^ 13 bytes) Physical memory size: 1 GB (2 ^ 30 bytes)

First, let's calculate the number of pages and frames:

Number of pages = Physical memory size / Page size

Number of frames = Physical memory size / Page size

a. Conventional, single-level page table:

In a conventional page table, each page entry corresponds to a page in memory.

Number of entries = Number of pages = Physical memory size / Page size = (1 * 1024 * 1024 KB) / (8 * 1024 bytes) = 128 meaning there are 128 entries in the conventional, single-level page table.

b. Inverted page table:

In an inverted page table, each entry corresponds to a frame in memory.

Number of entries = Number of frames = Physical memory size / Page size

Number of entries = (1 * 1024 * 1024 KB) / (8 * 1024 bytes)

Number of entries = 128, meaning there are also 128 entries in the inverted page table.