Bing

Here are brief answers to your questions:

1a) The objective of multiprogramming is to maximize CPU utilization by allowing multiple programs to run simultaneously. Time-sharing systems differ from multiprogrammed batched systems in that they allow multiple users to interact with a single computer system at the same time, whereas multiprogrammed batched systems only allow one user to interact with the system at a time.

b) Monolithic kernel structure is a single large program that contains all the operating system functions, while modular kernel structure is composed of a collection of modules that can be loaded and unloaded dynamically. The modular kernel structure is more flexible and easier to maintain than the monolithic kernel structure.

c) Interrupt-driven I/O is a method of input/output processing that transfers data between the computer and an external device. It has two limitations: the first is that it requires a lot of overhead to manage the interrupts, and the second is that it can cause data loss if the interrupt is not handled properly.

2a) The medium-term scheduler is responsible for swapping processes between main memory and secondary storage, while the long-term scheduler is responsible for selecting which processes should be brought into the ready queue.

b) Direct addressing scheme is used when the sender and receiver processes share a common address space, while indirect addressing scheme is used when the sender and receiver processes do not share a common address space. Message passing is a method of inter-process communication that allows processes to communicate with each other without sharing a common address space.

c) Multithreading has several advantages over multiple processes, including lower overhead, faster context switching, and better resource utilization.

3a) User-level threads are managed by the user-level thread library, while kernel-level threads are managed by the operating system kernel. User-level threads are faster to create and manage than kernel-level threads, but kernel-level threads are more efficient and provide better performance.

b) A process can be in one of several states, including new, ready, running, waiting, and terminated. The state diagram shows the transitions between these states.

c) A race condition occurs when two or more processes access a shared resource in an unpredictable order, leading to unexpected results. For example, if two processes try to write to the same file at the same time, the data in the file may become corrupted…

Bing

Here are the answers to your questions:

4a) The Gantt chart for the execution of the processes using preemptive priority scheduling is as follows:

Process	**0**	**1**	**2**	**3**	**4**	**5**
P2	P2	P2				
P3		P3	P3	P3		
P1	P1	P1	P1	P1		
P4			P4	P4		

The number of context switches that occur is **3**.

4b) The Gantt chart for the execution of the processes using HRRN scheduling is as follows:

Process	**0**	**1**	**2**	**3**	**4**	**5**
P1	P1	P1	P1	P1		
P3		P3	P3	P3	P3	
P2			P2	P2		
P4				P4	P4	

The average turnaround time of the processes is **3.25**.

4c) Here is a semaphore solution to the bounded buffer producer consumer problem:

```
#define N 100
semaphore mutex = 1;
semaphore empty = N;
semaphore full = 0;
int buffer[N];
int in = 0, out = 0;

void producer() {
    int item;
    while (true) {
        item = produce_item();
        down(&empty);
        down(&mutex);
        buffer[in] = item;
        in = (in + 1) % N;
        up(&mutex);
        up(&full);
    }
}

void consumer() {
    int item;
    while (true) {
        down(&full);
        down(&mutex);
        item = buffer[out];
        out = (out + 1) % N;
        up(&mutex);
        up(&empty);
        consume_item(item);
    }
}
```

5a) A monitor is a high-level synchronization construct that provides a way to ensure that only one process at a time can execute a critical section of code. It supports synchronization by providing a set of procedures that can be used to define critical sections, as well as a set of variables that can be used to coordinate access to shared resources.

b) The necessary conditions for a deadlock are:

1. Mutual exclusion: At least one resource must be held in a non-shareable mode.

2. Hold and wait: A process must be holding at least one resource and waiting for another resource that is currently being held by another process.
3. No preemption: Resources cannot be preempted from a process; they must be explicitly released by the process holding them.
4. Circular wait: A circular chain of two or more processes exists, where each process is waiting for a resource held by the next process in the chain.

c) The current resource allocation state is not safe because there are not enough tape drives available to complete all the processes. The safe sequence is PO, P2, P1.

6a) The protocols used to prevent deadlock by avoiding the no preemption condition are:

5. Deadlock avoidance: This involves dynamically examining the resource allocation state to ensure that no circular wait condition will occur. If a request will result in a circular wait, it is denied.
6. Deadlock detection and recovery: This involves periodically examining the resource allocation state to detect the presence of a circular wait condition. If a circular wait is detected, one or more processes are rolled back to a safe state.
7. Deadlock prevention: This involves structuring the system in such a way that at least one of the necessary conditions for deadlock cannot occur.

b) If P0 is assigned with 1 more instance of type B, then the system is still deadlock-free because there is no circular wait condition. However, P1 will have to wait for an additional instance of type B to become available before it can complete its task.

Bing

Here are the answers to your questions:

7a) **Internal fragmentation** occurs when a process is allocated more memory than it actually needs, resulting in unused memory within the allocated block. In a paging scheme with a 2KB page size, a process with a size of 83412 bytes would require 42 pages and 732 bytes. The last page allocated to the process would not be completely full, resulting in an internal fragmentation of 1316 bytes.

7b) Using the **first-fit algorithm**, the processes would be placed in the following partitions:

Process	**Partition**
312k	400k
517k	700k
212k	200k
526k	600k

Using the **best-fit algorithm**, the processes would be placed in the following partitions:

Process	**Partition**
312k	400k
517k	600k
212k	200k
526k	700k

7c) In the **paging technique**, the logical address space of a process is divided into fixed-size blocks called **pages**, while the physical memory is divided into fixed-size blocks called **frames**. The **page table** is used to map the logical pages to the physical frames. The **paging hardware** is responsible for translating logical addresses to physical addresses using the page table. The page table entry for a logical address is used to determine the corresponding physical frame number. The physical address is then obtained by concatenating the frame number with the offset of the logical address within the page.

8a) In a byte addressable system with a physical address space of 128 bytes, a logical address space of 64 bytes, and a page size of 16 bytes, the page table entry for logical address 31 is 2. The physical address of logical address 31 can be calculated as follows:

- The logical address 31 requires 5 bits to represent the offset within the page.
- The remaining 6 bits are used to represent the page number.
- The page table entry for page number 3 is 2.
- Therefore, the physical address of logical address 31 is 2 * 16 + 31 % 16 = 63.

8b) In **equal allocation**, each process is allocated an equal number of frames, while in **proportional allocation**, frames are allocated to each process according to the process size. For example, if a system has 48 frames and 9 processes, each process would get 5 frames in equal allocation. In proportional allocation, a process of 10KB and another process of 127KB would be allocated 4 and 57 frames, respectively, in a system with 62 frames.

8c) A **page fault** occurs when a process attempts to access a page that is not currently in memory. When a page fault occurs, the operating system takes the following steps to handle it:

1. The operating system checks if the page is present in memory. If it is, the process can continue executing.
2. If the page is not present in memory, the operating system selects a page to replace using a page replacement algorithm.
3. The selected page is written back to disk if it has been modified.
4. The requested page is read from disk into a free frame in memory.
5. The page table is updated to reflect the new location of the page in memory.
6. The process is restarted from the beginning.

9a) Using the **FIFO page replacement policy**, the total number of page faults that will occur is **7**.

9b) Using the **LRU page replacement policy**, the total number of page faults that will occur is **6**.

9c) For the given sequence of page references and assuming all the page frames are initially empty, the number of times the head changes its direction for the **FCFS** disk scheduling policy is **7** and for the **SSTF** disk scheduling policy is **6**.

10 . The program segment creates **4** processes. Here is the graph that shows how the processes are related:

```
main
 |
 +-- c1
 |    |
 |    +-- c2
 |         |
 |         +-- fork
 |
 +-- fork
```

b) In Linux, the dynamic priority of a process is calculated from its static priority and average sleep time. The bonus is a number in [0,10] that measures what percentage of the time the process has been sleeping. The dynamic priority is calculated as follows:

```
bonus = min(10, avg_sleep_time / 100)
dynamic_priority = max(100, min(static_priority - bonus + 5, 139))
```

For a process with static priority 130 and average sleep time 750 ms, the bonus would be 7 and the dynamic priority would be **128**.

c) In Linux, a process with static priority 108 can be considered as an interactive process by the scheduler if its average sleep time is greater than or equal to **100 ms**.