Consider a robot which is designed to move items around in a warehouse. The robot has a positioning system which combines input from its wheels (thereby allowing it to estimate its movements), from its cameras, as well as certain wireless beacons positioned around the warehouse. The robot can move freely within a specified area to perform its tasks, but if it goes outside the designated area a safety subsystem will immediately stop the robot.

The program controlling the robot is a parallel program and includes the following data structures.

- locations - This is a list of estimated <time, location> pairs. The list should not contain pairs that are too close to each other in time since this will result in too much overhead.
- particleFilter - Abstract data type that allows estimating the current position of the robot

The program also contains the following functions that can be called from the concurrent processes.

```
UpdateSensorReading(Sensor s, Data d) {
  //first update the particle filter with the new sensor data
  particleFilter.AddSensorData(s,d)
  //then estimate the current location
  EstimateCurrentLocation()
}

EstimateCurrentLocation() {
  //get last estimated location
  <t, loc> = locations.getLast()
  //if the last time stamp was sufficiently long ago
  if (now - t > threshold) {
    //update the location with current estimate
    newloc = particleFilter.GetLocation()
    locations.append(<now, newloc>)
  }
}
StopIfOutOfBounds() {
  <t,loc> = locations.getLast()
  if not InDesignatedArea(loc) {
    StopRobot()
  }
}
```

There are a number of sensor reading processes that are running concurrently, invoking the UpdateSensorReading() function. Moreover, the process below repeatedly invokes the safety check.

```
Process SafetyCheck {
  while (true) {
    StopIfOutOfBounds()
  }
}
```

P1 P2 P4] P2 P4 P4 P5 P2 P4

Question 1

None of the data structures, functions or processes running in the robot has any concurrency protection mechanisms enabled. Identify a potential race condition that can occur in the system and describe what the result could be. Motivate your answer.

Question 2

Add protection mechanisms in the above pseudo code (you can copy the code and paste it below, and then edit it) so that mutual exclusion of critical sections is ensured.

Question 3

The safety check uses a busy loop to make sure that the robot is stopped as soon as a new estimated location is available. However, it turns out that this loop is consuming a lot of processor capacity from the rest of the system. Use synchronization primitives (Semaphores or appropriately configured Condition variables) to stop the continuous loops.

Question 4

In your solution concurrency has been implemented on the high-level functions that control the robot behavior. An alternative approach in the general case is to make sure that the data types are thread-safe. For example, it is possible to construct a list class for which all operations can be safely called from concurrent processes by adding synchronization mechanisms in the list operations themselves.

Comment on the pros and cons of adding concurrency protection on higher-level functions (like the ones above), or on low-level functions (associated with basic data structures such as lists).

Question 5

Consider the following (Unix) C program.

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main() {
  pid_t pid1 = fork();
  if (pid1 == 0) {
    pid_t pid2 = fork();
    printf("A");
    pid_t pid3 = fork();
    if (pid3 == 0) {
      printf("A");
    }
  }
  printf("A");
}
```

How many times will the letter A be printed to the standard output when executing this program?

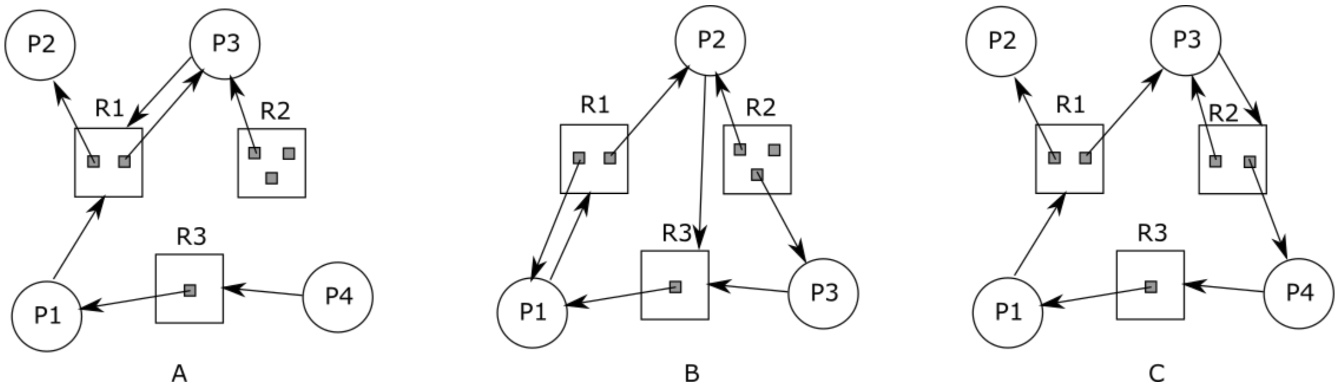
Question 6

The POSIX documentation contains the following information regarding the option WNOHANG which can be used for a variant of the *wait* function called *waitpid*.

"**WNOHANG**
The *waitpid*() function shall not suspend execution of the calling thread if *status* is not immediately available for one of the child processes specified by *pid*."

Describe a scenario where this option is needed and what problem it thereby avoids. Moreover, explain why the operating system cannot take care of this problem automatically.

Question 7



Consider three different systems, A, B, and C depicted above. Which of these systems are currently in a deadlock?

Select one or more:
a. A
b. B
c. C

Question 8

For each of the systems A, B, and C. Motivate why or why not the system is in a deadlock.

Question 9

Consider a single-CPU system and the following set of processes with arrival times (in milliseconds), expected maximum execution time (ms), and priority **(5 is highest, 1 is lowest priority)**.

Process	Arrival time	Execution time (ms)	Priority (as applicable)
P1	0	2	3
P2	1	1	2
P3	3	5	1
P4	5	3	5
P5	8	1	4

Assume that each slot below is a 1ms slot. Use the round-robin scheduling policy with a time quantim of 2ms to determine which process that executes in each slot. If a new process A arrives at the same time as another process B has just finished running in the CPU, A will be added first to the ready queue.

Question 10

Consider a single-CPU system and the following set of processes with arrival times (in milliseconds), expected maximum execution time (ms), and priority **(5 is highest, 1 is lowest priority)**.

Process	Arrival time	Execution time (ms)	Priority (as applicable)
P1	0	2	3
P2	1	4	2
P3	3	2	1
P4	5	3	5
P5	8	1	4

Assume that each slot below is a 1ms slot. Use the non-preemptive shortest job first scheduling policy to determine which process that executes in each slot. If a new process A arrives at the same time as another process B has just finished running in the CPU, A will be added first to the ready queue.

Question 11

Consider a single-CPU system and the following set of processes with arrival times (in milliseconds), expected maximum execution time (ms), and priority **(5 is highest, 1 is lowest priority)**.

Process	Arrival time	Execution time (ms)	Priority (as applicable)
P1	0	3	3
P2	1	4	2
P3	2	1	1
P4	5	3	5
P5	8	1	4

Assume that each slot below is a 1ms slot. Use the preemptive priority scheduling policy to determine which process that executes in each slot. If a new process A arrives at the same time as another process B has just finished running in the CPU, A will be added first to the ready queue.

Question 12

Question text

Consider a 64-bit architecture that uses 48bits to address logical memory. Assume that each page is 4kB and that a three-level paging is used. Each inner page table (must fit within one page) and each entry in a table is one word (64 bits = 8 bytes). What is the size of the outer page table (in kibibytes)?

Question 13

System designers in general and hardware manufacturers in particular are often pressed by market forces to produce systems with ever increasing performance. This applies also to the design of the memory management architectures. When evaluating the performance of two different solutions it is customary to use benchmark tests that are supposed to provide typical use-case characteristics and then to measure for example how long it takes to run a particular test case.

A consequence of this way of designing systems is that *average-case performance* is prioritized at the expense of *worst-case performance*.

Give two examples of mechanisms in memory management where the worst-case memory access time is increased in order to reduce the average-case memory access time. Moreover, explain under what circumstances (characteristics of programs) that the system is more likely to experience the worst-case effects.