Process communication, scheduling and memory management

1. What is a race condition? What is a deadlock?

    - race condition: when two processes are attempting to access the same resource simultaneously, with the result depending on exactly who accessed the resource first, and may produce unexpected/incorrect results
    - deadlock: when multiple processes are waiting for a signal from other processes, and none can proceed as a result.

2. In the lectures we saw that the priority inversion problem can happen with processes. Can the priority inversion problem happen with user-level threads? Why or why not?

- priority inversion problem: low, medium, high priority tasks L, M, H respectively.
  L and H use a shared resource R, accessing this in their critical region. M doesn't require access. L runs in critical region, H is waiting to run in critical region. H waits for L to come out of critical region. M interrupts L and starts running to completion. L resumes until completion of critical region, then H enters critical region and starts running. Here H was delayed by M, even though M has lower priority. This is priority inversion.
- Mars pathfinder failure
- Wikipedia: priority inversion
- user-level threads: kernel level threads get scheduled, kernel level sees single thread, so the priority inversion cannot happen

3. In lectures we saw a concept of a resource graph and a circular chain that leads to a deadlock. Give an example to show that the set of processes deadlocked can include processes that are not in a circular chain in the corresponding resource allocation graph.

- Consider a 3rd process that has obtained a mutex that is itself deadlocked

4. Can a measure of whether a process is likely to be CPU bound or I/O bound be determined by analysing source code? How can this be determined at run time?

- it will be hardware dependent so cannot be fully determined in advance
- could estimate (eyeball) based on relative proportion of heavy I/O statements to computation (e.g. writing 1M bytes will certainly time a lot of time)
- profiling probably useful for determining this. e.g. `time` provides user, real, & wall time
    - `strace`
    - bpf probes
    - `perf`

5. Five batch jobs, A through E, arrive at a computer centre at almost the same time. They have estimated running times of 10, 6, 2, 4, and 8 minutes. Their (externally determined) priorities are 3, 5, 2, 1, and 4, respectively, with 5 being the highest priority. For each of the following scheduling algorithms, determine the mean process turnaround time. Ignore process switching overhead.

(a) Round robin with quantum of 6 minutes.
(b) Priority scheduling.
(c) First-come, first-served (run in order 10, 6, 2, 4, 8).
(d) Shortest job first.

For (b) through (d), assume that only one job at a time runs, until it finishes. All jobs are completely CPU bound.

- mean process turnaround time: time from process start to completion

a) (12+14+18+30+32)/5 =
b) (6+14+24+26+30)/5 = 20
c) (10 + 16+ 18+22+30)/5 = 19.2
d) (2+6+12+20+30)/5 =14

6. Give an arithmetic expression connecting a given virtual address with the corresponding page number and offset. What does this tell you about page sizes that are not powers of two?

    - VA := virtual address
    - Y: num of offset bits
    - X: num of page bits
    - Page number = floor(VA/(page size); VA » Y
    - Offset = VA % (page size); VA & (1 « (X + Y) & 0 « X)
    - Page sizes that are not powers of two will not be able to be manipulated with bitwise operations (more expensive

7. How much physical memory do you need to store a page table describing

(a) 512 KB
(b) 4 GB given a page size of 512 bytes? How much do you need given a page size of 8 Kb? Assume that Page Table Entries (PTEs) are 4 bytes in either case.

Page size 512 bytes - a. 512KB = 512 * 1024 = 524288 Bytes = 1024 pages = 4kB - b. 4GB = 4294967296 bytes = 8388608 pages = 33554432 Bytes = 32 MB

Page size 8kB - a. 512kB / 8kB = 64 pages = 256 Bytes - b. 4GB / 8kB = 500000 pages = 2e6 bytes = ~2 MB

8. You are given the following data about a virtual memory system:

(a) The TLB can hold 1024 entries and can be accessed in 1 clock cycle (1 nsec).
(b) A page table entry can be found in 100 clock cycles or 100 nsec.
(c) The average page replacement time is 6 msec. If page references are handled by the TLB 99% of the time, and only 0.01% lead to a page fault, what is the effective address-translation time?

- TLB: translation lookaside buffer; cache of recently accessed page table entries
- TLB size: 1024 entries; 1 clock cycle access (1ns)
- PTE: found in 100 clock cycles

Effective address-translation time: 0.99*1ns + 0.01*100ns+0.0001 . 6ms = 602ns

9. In the lectures we saw that there are several strategies for choosing which free memory block (hole) to assign to a process. First/best/worst fit algorithms operate as follows: given a memory request of size X choose the first/smallest/largest hole that fits X, correspondingly. Consider a swapping system in which memory consists of the following hole sizes in memory order: 10 MB, 4 MB, 20 MB, 18 MB, 7 MB, 9 MB, 12 MB, and 15 MB. Which hole is taken for successive segment requests of (a) 12 MB (b) 10 MB (c) 9 MB for first fit? Now repeat the question for best fit, worst fit.

First fit: 20MB, 10MB, 18MB Best fit: 12MB, 10MB, 9MB Worst fit: 20MB, 18MB, 15MB

10. Working set page replacement algorithm will first try to evict pages that are not in the working set before evicting pages in the working set. Under what type of access locality does a working set algorithm perform best (i.e., minimise number of page faults) and why?

- Wikipedia: locality
- spatial locality: use of data elements within relatively close storage locations
- temporal locality: use of data/resources within a small time duration
- under spatial locality you minimise the number of page faults: temporal locality may reqiure many more pages to be stored, whereas spatial locality will mean that all spatially local pages are loaded
- different heuristics

Additional questions

11. Recall the multi-threaded web-server scenario during Week 9. Another design could be formulated via consumer-producer problem as follows. There is a dispatcher and a worker thread that both have access to a buffer of requests. The dispatcher writes a request to a buffer and the worker thread reads the request, removes it and serves it by returning the corresponding

page. If the buffer is empty, the worker sleeps until the dispatcher wakes it up. If the buffer is full, the dispatcher sleeps until worker wakes it up. See the pseudo-code below.

Under which circumstances this code can lead to worker and dispatcher sleeping at the same time?

Can you think of a way to resolve the problem?

```
#define N 100 /* number of slots in the buffer */
int count = 0; /* number of requests in the buffer */
void dispatcher(void) {
    int request;
    while (TRUE) { /* repeat forever */
        request = add_request( ); /* add next request */
        if (count == N) sleep(); /* if buffer is full, go to sleep */
        insert_request(request); /* put request in buffer */
        count = count + 1; /* increment counter */
        if (count == 1) wakeup(worker); /* was buffer empty? */
    }
}
void worker(void) {
    int request;
    while (TRUE) { /* repeat forever */
        if (count == 0) sleep(); /* if buffer is empty, go to sleep */
        request = remove_request(); /* take request out of buffer */
        count = count - 1; /* decrease the count */
        if (count == N - 1) wakeup(dispatcher); /* was buffer full? */
        serve_request(request);
    }
}
```

12. Consider a system with 32-bit virtual (logical) addresses, 24-bit physical addresses, and the system uses paging to manage memory. A page frame holds 4096 bytes. What is the maximum number of entries in the page table?

13. What is meant by the term thrashing?

    - when a program causes page faults every few instructions, indicating the working set is too small