Homework 2 Answers
May 9, 2022

When you do the homework, please put the answers to questions 1-4 on one page, and the answers to each of the others on separate pages. You can save this file and put your answers on it. This will make using Gradescope to grade the assignment much easier than if you submitted everything without regard to pages.

Remember, you must **justify all your answers**

Short Answer Questions
What are the four conditions that are necessary for a deadlock to occur?

Mutual exclusion, no preemption, hold and wait, and circular wait.

Consider a logical address space of 16 pages of 4096 words each, mapped onto a physical memory of 1024 frames. How many bits are there in the logical address? In the physical address?

The addresses must have enough space to hold the page or frame number, and the maximum offset within the page. So for the logical address, which has to support $16 = 2^4$ pages and offsets of up to $4096 = 2^{12}$, the addresses have $4 + 12 = 16\ bits$
The physical addresses must support $1024 = 2^{10}$ frames, each of which is $4096 = 2^{12}$ words long, so the address size is $10 + 12 = 22$ bits.

How does the Working Set replacement strategy relate process scheduling to memory management?

The working set page replacement strategy relates process scheduling to memory management by the Working Set Principle, which says that a process may run only when all pages in its working set are in memory, and pages in the working set of a the running process may never be replaced.

{Long Answer Questions}

Consider a system with three model airplane building processes and one agent process. Each building process requires a tube of glue, a piece of newspaper, and a model kit to put a model airplane together. One of the processes has tubes of glue, another piece of newspaper, and the third model kits. The agent has an infinite supply of all three. The agent places two of the ingredients for the model builders on the table. The process who has the remaining ingredient can then build one model airplane. It signals the agent upon completion. The agent then puts out another two of the three ingredients and the cycle repeats.

Write a program to synchronize the agent and the model building processes using monitors. Assume that if a process signals on a condition variable that another process is waiting on, the signaler blocks until the other process either leaves the monitor or blocks.

The easiest way to answer this is to realize that the agent is generating two items, and then in effect asking which builder has the one missing ingredient. This suggests the agent should signal based on the missing item, and each builder should wait for the signal on the item that it has. This observation leads to the following program, in which the routine function *build* is the building of the model and the routine function *random*(variable *low*, variable *high)* generates a random integer between variable *low* and variable *high* inclusive:

```
typedef enum things item = { nothing, glue, newspaper, model };
monitor {
condition notbuilding;         /* used when no builders are building */
condition ingredients[items]; /* used when the agent indicates what is
 missing */
items notontable = nothing;    /* the item not placed on the table;
     initially everything is
there */
int ready = 0;                /* number of builders ready to build */
/* put all but ingredient i on table; wait until builder is done */
entry void needed(item i) {
/* wait until all builders ready */
while (ready < 3)
notbuilding.wait;
/* put down materials */
notontable = i;
ingredients[i].signal;
}
/* builder takes it and others, starts building */
entry void take(item i) {
/* say you're ready */
ready++;
notbuilding.signal;
/* now wait */
if (notontable != i)
ingredients[i].wait;
/* going */
ready--;
notontable = nothing;
}
```

```
} build;                                /* name of the monitor */
/* builder -- it takes the missing item, builds, and finishes */
void builder(item hasitem) {
while (1) {
build.take(hasitem);
/* build airplane */
}
}
/* agent process -- it generates two items and puts them on the table; */
/* it does this by choosing one of the three items to be missing. */
void agent(void) {
item missingitem;
while (1) {
/* generate needed item randomly */
switch (random(0, 2)){
case 0: missingitem = glue; break;
case 1: missingitem := newspaper; break;
case 2: missingitem := model; break;
}
/* ask for it */
build.needed(missingitem);
}
}
}
void main(void) {
parbegin
agent;
builder(glue);
builder(newspaper);
builder(model);
parend;
}
```

The presence of the *while* loop at the top of the function *needed* seems unnecessary, but must be present to solve a synchronization problem when all four processes (the three builders and the agent) are started. If that loop were not present, the agent might enter the monitor many times (each time issuing a signal) before any builder enters its monitor. The agent must wait until all three builders are prepared to take something from the table, hence some sort of synchronization mechanism *wait* is needed.

The simplistic answer, 3 waits (one for each builder) fails as the agent might enter its monitor after the first two builders have entered theirs, but before the third builder enters its monitor. The presence of the counter variable *ready* ensures the wait in the loop is executed only as many times as necessary. The other point worth noting is the presence of the variable *notontable*.

When one uses Hoare's version of monitors, recall that after a signal is issued, the signaller blocks and one of the processes waiting for the signal resumes. If the variable *notontable* were not present, and the keyword *if* statements containing it were eliminated, the agent will occasionally not wait for what is placed on the table to be used before placing something else on the table.

For example: the builder with glue and newspaper finishes building and the agent is waiting at the variable *notbuilding*, keyword *wait* line. The builder reenters the monitor by calling the function *take* routine, increments ready, and signals on variable *notbuilding*. The agent now resumes because the signaller (the builder) suspends immediately after the signal).

The agent again places a model on the table and signals the builder with newspaper and glue. So the agent suspends, and the builder with newspaper and glue resumes (remember, suspended signallers have priority over processes trying to enter the monitor). It then blocks at the next keyword *wait* because the agent signaled it before it had blocked on the keyword *wait*. The agent now resumes, falls out of the monitor, re-invokes the procedure needed, and will signal again. That the agent had previously placed a model on the table, and that it was never used, is lost.

Assume that we have a paged memory system with a cache to hold the most active page table entries. It takes $20ns$ to search the cache. If the page table is normally held in memory, and memory access time is $1\mu s$, what is the effective access time if the hit ratio is 85%? What hit ratio will be necessary to reduce the effective memory access time to $1.1\mu s$?

Let h be the hit ratio. **Then the effective memory access time $EMAT$ is**

$$EMAT = h\,(1\mu s + 20ns) + (1 - h)(1\mu s + 20ns + 1\mu s) \text{ or, as}$$

$$1\mu\, s = 1000n\, s,\ EMAT = 1020h + 2020ns - 2020ns = 2020 - 1000h\, ns$$

Taking $h = 0.85$ gives $EMAT = 1170ns$

For the second part, we need to solve the above for s. This gives

$$h = \frac{2020 - EMAT}{1000}$$

so, reducing $EMAT\ to\ 1.1\mu\, s = 1100ns\ requires\ that\ h = 0.92,\ or\ a\ hit\ ratio\ of\ 92$

A virtual memory has a page size of 1024 words, 8 virtual pages, and 4 physical page frames. **The page table is as follows:**

Virtual Page	Page Frame
0	3
1	1
2	Not in Main Memory
3	Not in Main Memory
4	2
5	Not in Main Memory
6	0
7	Not in Main Memory

Which virtual addresses will cause page faults?
Virtual addresses between 2048 and 4095, 5120 and 6143, and 7168 and 8191 will cause page faults.

What are the physical addresses for 0, 3728, 1023, 1024, 1025, 7800, and 4096?

The required physical addresses are as follows.

Virtual Address	Page Address
0	3072
3728	Page Fault
1023	4095
1024	1024
1025	1025
7800	Page Fault
4096	2048