Short Answers

1. Mutual Exclusion, No Preemption, Circular Wait and Hold-and-Wait are the four conditions required for a deadlock to occur. Deadlocks occur when a needed resource is *never* available for allocation.
    a. Mutual exclusion is required because if multiple processes can gain access at the same time, then there is no competition of resources and no process waits on another. It can access the needed resource at any time.
    b. No pre-emption is required because if a scheduler or set of rules is able to detect the problem
    m ahead of time, it can force the resource to be moved from one process to another, breaking a deadlock.
    c. Circular Wait means that processes are blocked in a circular chain where each process holds a resource another needs but also relies on a resource from another. If Circular Wait is not a condition, then each process can finish their task and no deadlock occurs.
    d. Hold-and-Wait is the last condition with a similar premise to Circular Wait. If processes are not holding and waiting, then the resources will not be occupied or contested, preventing a deadlock from occurring.
2. Given that there are 16 pages in the logical address and the page size is 4 ($2^4 = 16$). Now there are 4096 words, then there are 2^{16} bits ($2^4 * 2^{12}$) in the logical address. For the physical address, given 1024(2^{10}) frames and for each word (2^{12}), then the total bits for the physical address is 2^{22} ($2^{10} * 2^{12}$).
3. The Working Set relates job scheduling to memory management through its principle. Its principle states that a process may execute only if its working set is resident in main memory. A page may not be removed from main memory if it is in the working set of the executing process. (2022-05-02-slides.pdf)
    a. Size of a working set can vary:
        i. $1 < |W(t, \tau)| < \min(\tau$, number of pages in process)
        ii. $W(t, \tau) \subseteq W(t+1, \tau)$ so this is a stack algorithm
        iii. Working set of a process undergoes periods of fairly consistent size alternating with periods of larger size
            1. Larger size (stable range) is when process is in locality
            2. Smaller size (transition range) is when process is transitioning from one locality to the next
        iv. Larger periods typically account for 98% of process time
        v. Remaining 2% has at least half of all page faults
        vi. Ideally,τ large enough so working set contains all pages being frequently accessed,and small enough so it contains only those pages.

Long Answers

Answer 4: Monitor Code

```
typedef enum twoItems = {modelGlue, modelNews, newsGlue}
monitor {
  condition modelGlue, modelNews, newsGlue
  condition completed

/* Builder tells what it needs and waits for signal */
  entry void needed(twoItems items){
    switch(items){
      case modelGlue:
        modelGlue.wait
        break;
      case modelNews:
        modelNews.wait
        break;
      case newsGlue:
        newsGlue.wait
        break;
    }
  }

/* Agent calls place to signal which two are available */
  entry void place(twoItems items){
    switch(items){
      case modelGlue:
        modelGlue.signal
        break;
      case modelNews:
        modelNews.signal
        break;
      case newsGlue:
        newsGlue.signal
        break;
    }
  }

/* Agent waits upon completion before sending out more */
  entry void waitComplete(){
    completed.wait()
  }

/* Builder signals when its completed its build */
  entry void signalComplete() {
    completed.signal())
  }
} build;
```

```
void builder(twoItems itemsNeeded) {
  /* builder starts with some random item, requests the other
two */
  while(1){
    monitor.request(itemsNeeded)
    /* Execute build here */
    monitor.signalComplete()
  }
}

void agent() {
  while(1){
    twoItems = newsGlue /* Random choice of the three */
    monitor.place(twoItems)
    monitor.waitComplete()
  }
}

void main() {
/* Any two/three items could be called in whatever sequence*/
  builder(twoItems.modelGlue)
  builder(twoItems.newsGlue)
  builder(twoItems.modelNews)
  agent()
}
```

Builder when called/created in main, will use the monitor to request resources and in doing so, will wait until it gets a response (from agent). Agent on the other hand will place two random items and signal those two items for any builder that comes along, once it has done so, it will wait until it gets the signal that something has been built/completed. Builder, once it receives the items it needs from its specific signal, will stop waiting and perform whatever operations it needs, then signals to agent that it completed. Agent and builders will do this continuously to build the planes.

Answer 5

Average time needed for a memory reference: hit ratio × time needed to reference page when page number in cache + (1 – hit ratio) × time needed to reference page when page number not in cache. (2022-04-29-slides.pdf)

Let $r = 20ns\ or\ 20 * 10^{-9}s$ or $.020 * 10^{-6}$ where r is the time it takes to reference a page number, $m = 1\mu s\ or\ 1 * 10^{-6}s$ where m is the memory access time, let $h = 0.85$ be the given hit ratio of 85% and let e be the effective time needed for a memory reference. Solve for e.

$$e = (h * r) + (1 - h)(r + 2m)$$

$$=> e = (0.85 * 0.20 * 10^{-6}) + (1 - 0.85)(0.02 * 10^{-6} + 2 * 10^{-6})$$

$$=> e = (0.17 * 10^{-6}) + (.15)(2.02 \& 10^{-6})$$

$$=> e = 32 * 10^{-6}s$$

Now solve for h to get the needed hit ratio such that:

$$1.1 * 10^{-6} = h(0.02 * 10^{-6} + 10^{-6}) + (1 - h)((0.02 * 10^{-6} + 2 * 10^{-6}))$$

$$=> h = 92\%$$

Answer 6

Part A: 1024 words per address x 8 virtual pages = 8192 virtual addresses

Virtual Page	Page Frame	Virtual Address
0	3	0-1023
1	1	1024-2047
2	Not in main memory	2048-3071
3	Not in main memory	3072-4095
4	2	4096-5119

5	Not in main memory	5120-6143
6	0	6144-7167
7	Not in main memory	7168-8191

Virtual addresses that will cause page faults are: 2048-4095, 5120-6143, 7168-8191. This is due to the fact that they are not in memory.

Part B: Using the formula of offset + page frame starting address

Virtual address is (segment number, segment offset). In this address, segment offset is (page number, page offset). Entries in segment table are (page table base, page table length) To get physical address from virtual address:

1. Get segment number and compare it to segment table length; if number greater, it’s an illegal reference

2. Add STBR to segment number

3. Get segment table entry

4. Add page number to page table base address

5. Get page table entry

6. Use the frame number in it and page offset to get physical address

Lecture Slides (2022-05-02-slides.pdf)

Virtual Address	Virtual Page	Physical Page	Physical Address
0	0	3	3072
3728	3	Page Fault	Page Fault
1023	0	3	4095
1024	1	1	1024
1025	1	1	1025
7800	7	Page Fault	Page Fault
4096	4	2	2048