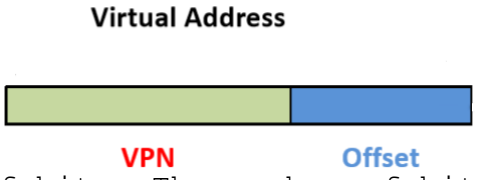
Stephen Woodbury – 1429496 – swoodbur
HW5 - CMPS 111 – Spring 2018 – 5/23/2018 - Wednesday

1) *Consider a computer with a 32-bit processor and 8 KB pages. Calculate the number of linear page table entries required if virtual addresses are 48-bit.*
A: A virtual address is made up of two parts, the virtual page number(VPN) and the offset.



Each is made up of bits. The number of bits in the offset determine how large your pages can be while the number of bits in VPN determine how many pages(linear page table entries) we can have. Before determining the number of pages we can have, we need to determine the offset.

We know that a page is 8KB or 8192 Bytes. To address each Byte of our 8192 Bytes, we would need 13 bits to uniquely identify each Byte.

2^13 = 8192

That leaves 35 bits of our 48 bit virtual address leftover to determine the number of pages we can have.

2^35 = 34,359,738,370

This means that there are 34,359,738,370 linear page table entries required.

Stephen Woodbury – 1429496 – swoodbur
HW5 - CMPS 111 – Spring 2018 – 5/23/2018 - Wednesday

2) *A system has four processes and five allocatable resources. The current allocation and maximum needs are as follows:*

	Allocated	*Maximum*	*Available*
Process A:	*1 0 2 1 1*	*1 1 2 1 4*	*0 0 X 1 1*
Process B:	*2 0 1 1 1*	*2 2 2 1 0*	
Process C:	*1 1 0 1 0*	*2 1 3 1 0*	
Process D:	*1 1 1 1 1*	*1 1 2 2 1*	

Calculate the smallest value of X for which this is a state from which most processes can run to completion without deadlock. Show your work and explain your answer.

A: Couple of notes before beginning.

Process B currently has Resource 5 allocated to it despite it needing a maximum of 0 Resource 5's. I will take this to mean that the maximum is merely a guide for what the maximum number of resources said process could need but that it is not a definite maximum. That being said, for addressing this problem, I will assume that the despite Process B violating the maximum resources needed estimate, that the Maximum resources needed for each process is fairly accurate and that violations of the maximum are extremely rare.

I am also assuming that Process B will not drop it's one copy of Resource 5. This does in fact change our answer. If you want to know what X is given Process B lets go of Resource 5 immediately after acquiring it, look to the end of this answer.

Let's first look at the the leftover resources available:
R1: 0 R2: 0 R3: X R4: 1 R5: 1

Now let's look at what resources processes could potentially still need (maximum-allocated)

PA	PB	PC	PD
R1: 0	R1: 0	R1: 1	R1: 0
R2: 1	R2: 2	R2: 0	R2: 0
R3: 0	R3: 1	R3: 3	R3: 1
R4: 0	R4: 0	R4: 0	R4: 1
R5: 3	R5: (-1)	R5: 0	R5: 0

Based on the above information, only process D can execute next safely given there is at least 1 resource D available.
This is because There isn't enough R5 to allow PA to run safely.
There isn't enough R2 to allow PB to run safely.
There isn't enough R1 to allow PC to run safely.
There is enough of R4 to allow PD to run safely given we have 1

R3 available to cover the max R3 PD can use.

So, at the moment, X is at least 1.

Let's look at the next step, let's assume PD ran and has completed, freeing up it's taken resources. Let's take a look at the resources now available.
R1: 1 R2: 1 R3: X+1 R4: 2 R5: 2

Let's also look at Now let's look at what resources processes could potentially still need (maximum-allocated) again:

PA	PB	PC
R1: 0	R1: 0	R1: 1
R2: 1	R2: 2	R2: 0
R3: 0	R3: 1	R3: 3
R4: 0	R4: 0	R4: 0
R5: 3	R5: (-1)	R5: 0

Based on our updated information, Process C is the only process that can be ran safely given there are at least 3 R3's available.
This is because there isn't enough R5 for PA to run safely.
There isn't enough R2 for PB to run safely.
There is enough R1 for PC to run safely given there are 3 R3's available.

Using simple arithmetic, this means X has to be at least 2.
X+1=3 : X >= 2.

From here on out, whether our remaining processes deadlock or not is independent of Resource 3 given X=2. This is because our remaining processes need less of Resource 3 than is available. Given Process C completes, there will be at least 3 copies of resource 3 available for Process B and Process A to use in addition to what they already have allocated to themselves. Given that Process A doesn't need anymore R3 and that Process B needs only one more R3 at most, Resource 3 no longer determines if a deadlock will occur or not.

Therefore, X=2.

Extra Notes: Give PB drops R5 immediately, X=1. This is because after PD runs, there will be 3 copies of R5 available instead of 2, this allows PA to run safely, which, given it completes, will provide more than enough R3 for everything else to run.

Stephen Woodbury – 1429496 – swoodbur
HW5 - CMPS 111 – Spring 2018 – 5/23/2018 - Wednesday

3) *Can two kernel-level threads in the same user-level process synchronize using a kernel-level semaphore? What if the threads are implemented entirely at user-level? Assume that no threads in any other processes have access to the semaphore. Discuss your answers.*

A: If our threads are handled and implemented using the kernel, this means our kernel is aware there are two different threads running for one process. When one thread enters a critical section guarded by the kernel-level semaphore (effectively locking the critical section), our other thread will block on the semaphore if it tries entering the same critical section. Despite one of threads blocking, the process overall will keep running (again, assuming our kernel implemented the threads). This is because our kernel knows that there are multiple threads running for the same process and that if one thread blocks, another thread can be ran. So yes, two kernel-level threads in the same user-level process synchronize using a kernel-level semaphore.

If we look at two threads defined instead in our user-space however, we will find that our two threads cannot synchronize using a kernel-level semaphore. This is because out kernel is not aware that there are multiple threads, it only sees one process(the kernel being unaware that the process is managing its own threads). When a user-level thread locks the semaphore, the other user-level thread will block on the semaphore given it tries to access the same critical section. However, the kernel will think that the process as a whole blocks, which it will in turn stop the process from running.

Stephen Woodbury – 1429496 – swoodbur
HW5 - CMPS 111 – Spring 2018 – 5/23/2018 - Wednesday

4) *Early computers did not have Direct Memory Access (DMA); the CPU handled every byte of data read or written. (a) Briefly describe the mechanism by which these early computers handled non-DMA read operations. (b) What impact did the lack of DMA have on multiprogramming? Use diagrams if you feel they make your answers clearer*

A:

a) Without the DMA, the CPU does the I/O from disk. This bogs the CPU down immensely.

b) Without DMA, the CPU must handle the disk reading and writing. This means that the CPU cannot be working on another process while it's doing I/O(It can work on another process with a DMA). This makes the CPU act like it would in a system without multi-programming.

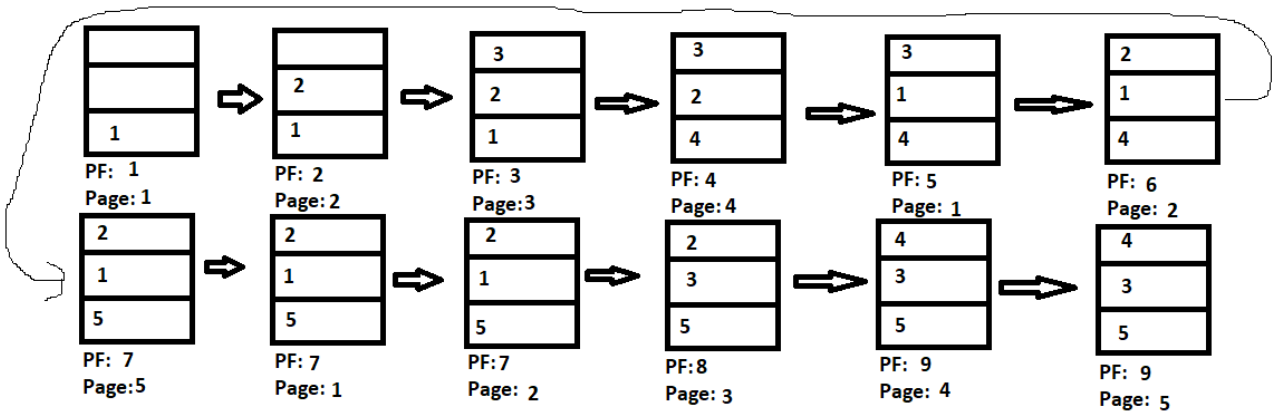
Stephen Woodbury – 1429496 – swoodbur
HW5 - CMPS 111 – Spring 2018 – 5/23/2018 - Wednesday

5) *Consider the FIFO page replacement algorithm and the reference string:*

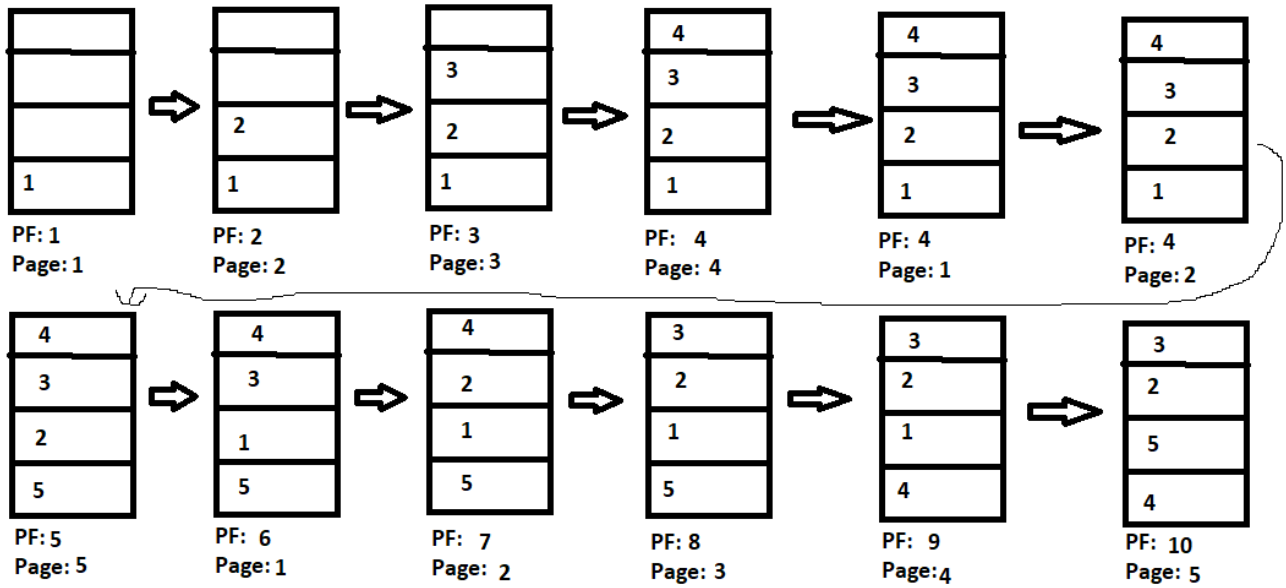
1 2 3 4 1 2 5 1 2 3 4 5

If the number of page frames increases from three to four, does the number of page faults go down, stay the same, or increase? Explain you answer using diagrams as appropriate.
A: The number of page faults goes up.

Three Frames servicing our pages:



Four Frames servicing our pages:



NOTE: PF: # of page faults; Page: Page being serviced

With four frames, there are ten page faults while with three frames, there are nine page faults.