0 with probability 1/4, X = 1 with pr

ment generating function of X?

b. $1/2 + 3e^{2t}/4$. c. $1/4 + 2e^t + 3e^t/4$. d. $1/4 + e^t/2 + e^{2t}/4$ e. None of

that you will have a straight flush on the turn?

like the case where you have 10♥ 9♥ and the board is

e. None of the above.

1. In a system using modern virtual memory techniques, what is the relationship between a page and a page frame?

A page is the virtual memory that is mapped to a segment of physical memory. Since a page can be in either memory or disk, the page frame is the memory a page is allocated.

2. Why are operating system ABIs of importance for convenient application software distribution?

ABI's are portable and robust, since they are guaranteed to work on a certain platform. Since they don't need to be compiled and can just be executed, it makes it easier for users to use since they won't have to deal with compilation issues

3e^t/4.  b. $1/2 + 3e^{2t}/4$.  c. $1/4 + 2e^t + 3e^t/4$.

...bility that you will have a straight flush on the turn?
...outcomes like the case where you have 10♥ 8♥ and the board is

...0001%.  e. None of the above.

7. Why is blocking a problem for user-mode threads? Why isn't it a problem for kernel-mode threads?

Since the OS is unaware of user-mode threads, it is unable to include user-mode threads in ~~preemptive scheduling~~ interrupts to switch threads. This isn't a problem for kernel-mode threads since the operating system is aware of them. Thus, the OS is able to context switch kernel-mode threads with interrupts, which allows for preemptive scheduling. It also allows kernel-mode threads to run on multiple cores.

8. Why does Shortest Time-To-Completion First (STCF) scheduling require preemption?

When a new process is queued, STCF ~~calculates~~ knows the time to completion. If the new process has a shorter time to completion than the currently running process, STCF will interrupt and context switch the current process for the shorter, newer one.

0 with probability 1/4, X = 1 with pr
ment generating function of X?
b. 1/2 + 3e^{2t}/4. c. 1/4 + 2e^t + 3e^t/4. d. 1/4 + e^t/2 + e^{2t}/4. e. None

robability that you will have a straight flush on the turn?
nt outcomes like the case where you have 10♥ 8♥ and the board is

9. When a Unix-system follows a fork with an exec, what resources of the forked process are replaced?

Almost everything is replaced, such as data, code segment, heap, ~~and the~~ whole address space ~~forked process not taking~~ ~~[illegible] process is and also the address space~~?

Since everything is copy on write, these ~~[illegible]~~ resources are ~~only written to~~ copied and the changes are written after the ~~[illegible]~~ exec. A fork and an exec essentially create a blank process. IPC is usually not replaced as well as file descriptors.

10. What form of fragmentation do we still suffer if we use a paging memory management system? For a se~~gmented pag~~ing system, how much fragmentation per segment do we see? ~~lots~~ fixed partition

Paging management system sees minor internal fragmentation since ~~sometimes~~ a page frame is a fixed size and so the memory needed could be less than that size. It ~~[illegible]~~ does not suffer significantly from external fragmentation since paging solves the need for contiguous memory. ~~For a segmented paging system, we see around 50% of fragmentation.~~

A fixed sized partitioning system will not suffer from external fragmentation, ~~as [illegible]~~ since the partitions of memory will fit well, but it suffers heavily from internal fragmentation. This is because a ~~process~~ program will be allocated a partition that fits the ~~process~~ program with the least extra memory. Therefore on each memory allocation, the program is usually allocated more memory that it asked for.

se $X = 0$ with probability 1/4, $X = 1$ with pro
he moment generating function of X?
/4. b. $1/2 + 3e^{2t}/4$. c. $1/4 + 2e^t + 3e^t/4$. d. $1/4 + e^t/2 + e^{3t}/4$. e. None of

bability that you will have a straight flush on the turn?
outcomes like the case where you have 10♥ 8♥ and the board is
001%. e. None of the above.

3. Why is information hiding a good property in an operating system interface?

~~Any operating system is extremely complicated so details are not important about how something~~

An operating system should include transparency and protection. Transparency allows for easier & efficient implementation, so ~~abstracting the~~ for example, abstracting memory allows a process to ~~think~~ ~~its~~ to not worry about ~~the~~ other processes. Information hiding also allows Protection, so a process won't be able to step on other processes or compromise them. Allowing a process to see memory of another process violates privacy and is dangerous if it can access it. Thus information hiding allows for easier implementation and also safety.

4. When an operating system performs a context switch between processes, what information must the OS save?

The OS must save the registers in the kernel stack and other information about the process state. This includes the program counter and ~~[illegible]~~ PS. This way, when this process is run again, it can resume execution with the same state with good correctness. Caching for a context switch is not saved, which is a huge performance cost.

= 0 with probability 1/4, X = 1 with p

oment generating function of X?

b. $1/2 + 3e^{3t}/4$. c. $1/4 + 2e^t + 3e^t/4$. d. $1/4 + e^t/2 + e^{2t}/4$ e. None

will have a straight flush on the turn?

like the case where you have 10♥ 8♥ and the board is

01%. e. None of the above.

5. What is the purpose of a trap table?

The trap table contains information that maps a trap type# (exception number) to its handler. It is initialized at boot time. Traps are important since it allows processes to run privileged instructions, and also allow for exception handling so that the OS is robust.

6. What is a race condition?

It is when the output of a program is dependent on the order of which instructions are run. This is due to ~~context switching~~ to ~~from~~ preemptive scheduling or if processes are running concurrently on separate cores. An example is when two threads are incrementing a shared variable. Since incrementing it requires loading and adding and then storing, if a thread loads and adds the shared variable before saving it, and the other thread runs, it has the copy of the non incremented variable. We solve race conditions with locks/ atomic instructions.