NAME Evan Bonsignori

Student-ID
P.ID: 907190445

Q1
OS
Sept. 6, 2018
Time: 45 minutes

1- At the machine cycle level, provide **step-wise** explanation of the activities that takes place before, during, and after an "interrupt". (10 pts.)

1. Fetch: fetches instruction that PC (program counter) points to and stores it in IR (instruction register).
2. Decode: Meaning of instruction is decoded
3. Execute: Instruction is computed
4. If IRQ (interrupt request) flag is set to true (because an interrupt was fired), then the interrupt is handled through context switching, otherwise steps 1-3 are repeated with an incremented PC.

+10

Context Switching:

5. First, all of the registers contents are stored in old state.
6. The PC is given the address of the solution stored in the interrupt vector. Corresponds to the interrupt fired, we will call this interrupt #, IRQ#, and the vector contains solutions IV (inter
7. After the solution is executed, the contents of old s are loaded into the registers, restoring the context before interrupt and the next machine cycle occurs.

Pseudo Code.

```
IR = fetch (PC)
// decode
execute (IR)
if (IRQ) {
```

```
OldState = saveRegisterContents()
restorePC (IV(IRQ#))
restorePC (OldState)
```

2. Define "OS", "Memory Mapped I/O", and "BIOS" (10 pts.)

+9.5

O.S.: A collection of programs and data that are in charge of managing computer resources, and supporting virtual characteristics of a computer. +3

Memory Mapped I/O: A device controller contains registers containing the instructions it is to perform. Typically, the CPU populates those registers with the instruction - the first register w/ the type, and the rest w/ information. To speed up this process, we use (as an example.) memory mapped I/O. This means we emulate a device controller register in RAM and create the illusion from the perspective of the device controller that it has physical registers. When the CPU wants to send instructions to the controller, it simply passes the address of the first byte of the emulated registers in the RAM. +4

BIOS: Bios is the bootloader software that is included in the ROM of a PC. It is upgradable, implemented Shadowly and is loaded at startup to ~~initialize~~ locate & load the Kernel into the RAM and load the essential components needed for a Computer to run. It is stored on the motherboard of

+2.5

not clear +0.5

+6

3- How (3 pts.)
a. The parameters needed for execution of a system call are communicated with an OS?
b. The messages between two processes are communicated with each other?

(a) Using one of three methods:

1. They are saved in registers before the call, and the OS extracts them / User them when the call is made.
2. They are stored in memory (RAM). This is the preferred method. predefine place in
3. They are pushed to a stack. +3

(b) Using of of two methods:

1. Shared Memory Space:
   Communications are sent to a memory space shared by process A and B.

+3

2. Message Passing:
   Communication are sent and retrieved from a dedicated space in the OS by processes.

6.5

4- Compare (itemize only the differences) (3 pts. each):
a- "Interrupt" and "trap"
b- "User mode" and "System mode"
c- "Loosely coupled" and "Tightly coupled" multi-processor systems
d- "Linker" and "loader"

Note: For each comparison, make sure that you do not leave the comparison to me by only providing the definitions of the two items (you lose points). Itemize only the differences between the two concepts and do not give me their definitions. You could start as follows:

The differences are: 1- ....., 2- ....., 3- ....., etc.

a) 1. Interrupt signal fired by OS vs. Signal fired by process
2. Interrupt given straight to OS vs. Signal is first handled by process (if a solution exists) otherwise sent to OS to do what
3. Checked against 2 separate solution vectors (interrupt vector & trap ve...

×2.5

b) 1. System mode: Interrupts/Traps not accepted (allow to queue)
User mode: Interrupts/Traps can be fired and get attention of OS.

2. System mode: process is "trapped" in monitor mode
User mode: process is changed and can finish execution

+0.5

3. System mode: when mode bit = 1
User mode: when mode bit = 0

c) 1. tightly coupled share CPU clock cycle vs loosely does not
2. tightly share system bus vs loosely does not
3. tightly may share memory, peripherals, I/O devices, & power supplies vs not loosely
4. loosely coupled nodes?? in separate machines communicating through ?! Ethernet/telephone cables vs tightly communicating within system through bus

+2.5

5. tightly couple is more fault tolerant (can fail back on shared hardware) cost effective (don't have to reboot shared components), and has a higher throughput (more CPUs) than loosely coupled system

1. Loader ~~loads program into RAM~~ ~~executable module and loads it into RAM~~, but does not save a copy on disk v.s. linker, creates executable module ~~and loads it to RAM~~, but saves a copy of the executable module on disk.

2. Linker combines the machine code of libraries into a ~~executable module, but loader keeps them separate.~~