6.004 Computation Structures
Spring 2009

Operating system issues

Problem 1. The following code outlines a simple timesharing scheduler:

```
struct MState {
  int Regs[31];                 /* saved state of user's registers */
} User;

int N = 42;                     /* number of processes to schedule */
int Cur = 0;                    /* number of "active" process */

struct PCB {
  struct MState State;          /* processor state */
  Context PageMap;              /* VM map for process */
  int DPYNum;                   /* console/keyboard number */
} ProcTbl[N];                   /* one per process */

Scheduler() {
  ProcTbl[Cur].State = User;    /* save current user state */
  Cur = (Cur + 1)%N;            /* increment modulo N */
  User = ProcTbl[Cur].State;    /* make another process the current one */
}
```

Suppose that each time the user hits a key on the keyboard, an interrupt is generated and the interrupt handler copies the new character into a kernel-resident input buffer. The operating system includes a ReadKey service call (SVC) which can be invoked by the user to read the next character from the input buffer. If the input buffer is empty, the SVC should "hang" until a character is available.

The first draft of the ReadKey SVC handler is shown below. The SVC handler routine saves the user's state in the User structure and then call ReadKey_h(). When ReadKey_h() returns, the SVC handler restores the user's state and then does a JMP(XP) to restart the user's program.

```
ReadKey_h() {
  int kdbnum = ProcTbl[Cur].DPYNum;
  while (BufferEmpty(kdbnum)) {
    /* busy wait loop */
  }
  User.Regs[0] = ReadInputBuffer(kdbnum);
}
```

A. ★ After executing a ReadKey SVC, where will the user program find the next character from the input buffer?

B. ★ Explain what's wrong with this proposed implementation.

C. ★ A second draft of the keyboard handler is shown below:

```
ReadKey_h() {
  int kdbnum = ProcTbl[Cur].DPYNum;
  if (BufferEmpty(kdbnum))
    User.Regs[XP] = User.Regs[XP] - 4;
  else
    User.Regs[0] = ReadInputBuffer(kdbnum);
}
```

Explain how the modifications fix the problems of the initial implementation.

D. ★ The designers notice that the process just wastes its time slice waiting for someone to hit a key. So they propose the following modifications:

```
ReadKey_h() {
  int kdbnum = ProcTbl[Cur].DPYNum;
  if (BufferEmpty(kdbnum)) {
    User.Regs[XP] = User.Regs[XP] - 4;
    Scheduler();
  } else
    User.Regs[0] = ReadInputBuffer(kdbnum);
}
```

What would be the most likely effect of removing the line of code where User.Regs[XP] is decremented by 4?

E. ★ Explain how this modification improves the overall performance of the system (assume the decrement by 4 has not been removed).

F. ★ This version of the handler still doesn't prevent the process from being scheduled each quantum, even though it may just call Scheduler() once again if no character has appeared in the input buffer. Explain how the sleep(status) and wakeup(status) kernel routines described in lecture can be used to make sure that processes are only scheduled when there is something "useful" for them to do.

Problem 2. The following questions are about the simple timesharing kernel used in Lab 8. Click here to view the code in a separate window.

A. ★ What happens right after reset?

B. ★ How does the processor get to execute the other user processes?

C. ★ How do supervisor calls (SVCs) work?

D. ★ How do characters typed on the keyboard find their way to user-mode programs?

Problem 3. Real Virtuality, Inc markets three different computers, each with its own operating system. The systems are:

Model A: A timeshared,multi-user Beta system whose OS kernel is uninterruptable.

Model B: A timeshared Beta system which enables device interrupts during handling of SVC traps.

Model C: A single-process (not timeshared) system which runs dedicated application code.

Each system runs an operating system that supports concurrent I/O on several devices, including an operator's console including a keyboard. Les N. Dowd, RVI's newly-hired OS expert, is in a jam: he has dropped the shoebox containing the master copies of OS source for all three systems. Unfortunately, three disks containing handlers for the ReadKey SVC trap, which reads and returns the ASCII code for the next key struck on the keyboard, have gotten confused. Of course, they are unlabeled, and Les isn't sure which handler goes into the OS for which machine. The handler sources are

```
ReadCh_h() {                              /* VERSION R1 */
  if (BufferEmpty(0))                     /* Has a key been typed? */
    User->Regs[XP] = User->Regs[XP]-4;    /* Nope, wait.  */
  else
    User->Regs[0] = ReadInputBuffer(0); /* Yup, return it. */
}

ReadCh_h() {                              /* VERSION R2 */
  int kbdnum=ProcTbl[Cur].DpyNum;
  while (BufferEmpty(kbdnum)) ;           /* Wait for a key to be hit*/
  User->Regs[0] = ReadInputBuffer(kbdnum);  /*...then return it. */
}
```

```
ReadCh_h() {                                /* VERSION R3 */
  int kbdnum=ProcTbl[Cur].DpyNum;
  if (BufferEmpty(kbdnum)) {                /* Has a key been typed? */
    User->Regs[XP] = User->Regs[XP]-4;  /* Nope, wait. */
    Scheduler();
  } else
    User->Regs[0] = ReadInputBuffer(kbdnum);    /* Yup, return it. /
}
```

A. Show that you're cleverer than Les by figuring out which handler goes with each OS, i.e., for each operating system (A, B and C) indicate the proper handler (R1, R2 or R3). Briefly explain your choices.

B. But Les isn't that clever. In order to figure out which handler code goes with each OS version, Les makes copies of each disk and distributes them as "updates" to beta-test teams for each OS. Les figures that if each handler version is tried by some beta tester in each OS, the comments of the testers will allow him to determine the proper OS for each handler.

   Les sends out the alleged source code updates, routing each handler source to testers for each OS. In response, he gets a barrage of complaints from many of the testers. Of course, he's forgotten which disk he sent to each tester. He asks your help to figure out which combination of system and hander causes each of the complaints. For each complaint below, explain which handler and which OS the complainer is trying to use.

   Complaint: "I get linkage errors; Scheduler and ProcTbl are undefined!"

C. Complaint: "I can link up the system using the new handler, but the system hangs when my application tries to read a key."

D. Complaint: "Hey, now the system always reads everybody's input from keyboard 0. Besides that, it seems to waste a lot more CPU cycles than it used to."

E. Complaint: "Neat, the new system seems to work fine. It even seems to waste less CPU time than it used to!"