Fall Semester 2011, 22C:113 Final Exam

Thursday 15 December 2011 7:30am–9:30am, Room: 51 SH

Instructions:

The maximum number of points is 200. To contest for maximum credit you are required to solve two of the following three problems. Please note that Problem I is mandatory, hence you must solve Problem I and one of Problem II or Problem III.

Problem I: Reliability, Efficiency, and Convenience of the BOS (100 points)
The action TSS below represents the evolution of the control program to a reliable, efficient, and convenient operating system called Time-Sharing System.

```
TSS::
   inout DeviceBuffer, CommandQueue;
   repeat
      GetCommand(Device,DeviceBuffer);
         ||
      EnqueueCommand(DeviceBuffer,CommandQueue);
         ||
      ExecuteCommand(CommandQueue);
   forever
```

The `semaphore` was defined as a lock data-type used to implement the interaction between the three main components of the time-sharing system: `GetCommand()`, `EnqueueCommand()`, `ExecuteCommand()`. You are required to answer the following questions:

1. (20 points) What is the data-carrier of a variable of type semaphore? (5 points). The three main operations which define the semaphore data type are `Set()`, `Signal()`, `Wait()`. Assuming that S is a variable of type semaphore explain the meaning of operations Set(S,Expression), Signal(S), Wait(S) (do not forget that they are atomic). You get 5 points for each of these operations correctly specified.

   Solution sketch:

   (a) Data-carrier of a variable of type semaphore is the set of natural numbers $\mathcal{N}$ (including 0), i.e., $\mathcal{N} = \{0, 1, 2, \ldots\}$.

   (b) $Set(S, Expression)$ means:
   Perform atomically the statements: $V := Value(Expression)$;
   $Signal(S)$ means: perform atomically the statement $S := S + 1$;
   $Wait(S)$ means: perform atomically the statement $S := S - 1$.
   Note, if $S = 0$ $Wait(S)$ implies delays because $0 - 1$ is not a natural number.

2. (25 points) Explain how is a semaphore variable S used for process synchronization (5points). Illustrate your explanation showing how are the three processes `P1:`

`GetCommand(), P2: EnqueueCommand(), P3: ExecuteCommand()` (that represent the core of a time-sharing system) synchronized using semaphore variables (20 points).

Solution sketch: (1) a semaphore variable S can be used to synchronize the actions performed by two processes p and q such that p performs the action A_2^2 only after q has performed the action A_1^1 by setting the initial value of S to 0 and designing the code performed by q and q as follows:
p: L1: action A_1^1; L2: Signal(S); L3: action A_2^1
q: L1: action A_1^2; L2: Wait(S); L3: action A_2^2

Example: Let P_1 perform the function `GetCommand()` and P_2 perform the function `EnqueueCommand()`. Then P_2 can send a command in the Time-sharing command queue Q only after that command was read by P_1 in the command buffer B. Further, assume that P_3 is the process performing the function `ExecuteCommand()`. Then P_3 can be forced to execute a command form Q only after that command has been sent by P_2 into the command queue Q. In conclusion, a Time-sharing system needs to use two semaphores variables, `Command` and `Execute`, initialized to 0 and P_1, P_2, and P_3 need to be designed as follows:

```
P1: GetCommand (Device, B); Signal(Command);
P2: Wait(Command); Enqueue(B,CommandQueue); Signal(Execute);
P3: Wait(Execute); ExecuteCommand(NextCommand(CommandQueue));
```

3. (25 points) Explain what does it mean that two processes share a resource in mutual exclusion (5 points). Show the structure of a critical-section of two processes p and q that share a resource R (5 points). Explain the resources `P1, P2, P3` share (5 points) and show their critical sections while sharing these resources (10 points).

   Solution sketch: two processes P_1 and P_2 share a resource R in mutual exclusion when only one process operates on R at a given time. The mutual exclusion of processes P_1 and P_2 operating on a shared resource R is implemented by a semaphore variable S initialized to 1 where the codes of P_1 and P_2 has the structure:

   P1: UsualCode1; Wait(S); P_1 operates on R; Signal(S); UsualCode2
   P2: UsualCode3; Wait(S); P_2 operates on R; Signal(S); UsualCode4

   The three processes that make up a time-sharing system share the buffer B where a command is read and the command queue Q where commands are waiting for execution. Hence the pseudo-code performed by the processes `P1: GetCommand(), P2: EnqueueCommand(), P3: ExecuteCommand()` use a semaphore `Buffer` initialized to 1, that allows `P1, P2` to share the buffer B in mutual exclusion, and a semaphore `Queue` initialized to 1, that allows `P2, P3` to share the command queue Q in mutual exclusion.

4. (30 points) Let again `P1, P2, P3` be the three processes that make up the core of the TSS, where `P1` performs `GetCommand()`, `P2` performs `EnqueueCommand()`, and `P3` performs `ExecuteCommand()`. Since `P1, P2, P3` synchronize their actions and share resources, the design of the TSS requires three parts:

   (a) A common part, where their synchronization and sharing semaphores (`Command, Execute, Buffer, Queue`) are declared and initialized, along with their shared variables (Input buffer B, and command queue Q);

   (b) A global part where the parallel actions performed by these process are executed;

   (c) The code performed by each individual process `P1, P2, P3`.

   You are required rewrite the action TSS showing the structure of a time-sharing system in terms of the three components mentioned above. You get 10 points for each component correctly designed.

   Solution sketch:

```
Time Sharing System:
Common part:
  semaphore Command, Execute, Buffer, Queue;
            Set(Command, 0); Set(Execute, 0); Set(Buffer,1); Set(Queue,1);
  buffer B;
  commandQueue Q;

Global action:
  Repeat P1 || P2 || P3 Forever

Individual actions:
  P1: GetCommand:
       Wait(Buffer) Read(Device,B);Signal(Buffer);Signal(Command);
P2: EnqueueCommand:
       Command C;
       Wait(Command);
       Wait(Buffer); C := B; Signal(Buffer);
       Wait(Queue) EnqueueCommand(C,CommandQueue); Signal(Queue);
       Signal(Execute);
ExecuteCommand:
       Command C;
       Wait(Execute);
       Wait(Queue);C:=GetNextCommand(CommandQueue);Signal(Queue);
       Perform(C);
```

Problem II: Programming Support Environment, PSE (100 points)

The PSE of a computer platform is a collection of system software tools that provide services to computer users for program development on a given computer platform. Please answer the following questions regarding these tools:

1. (10 points) Give five examples of software tools that belong to the PSE of a Linux machine. For each example indicate the service(s) it performs.

   Sketch of answer:

   (a) A control language, allows computer users to give commands to the software tools available in the PSE.

   (b) A control language interpretor, maps control-language commands into processes executing them.

   (c) Compiler, maps high-level /Solution programming languages into machine languages. Compilers allow programmers to develop programs using a human oriented logic while programs are executed by the machine using a machine oriented logic

   (d) Debugger, runs programs under the control of the programmer. Debuggers allow programmers to check the correctness of their programs by "step by step execution" in conditions determined by programmer according to program behavior.

   (e) Editor, creates and update files. Editors allow programmers to write their programs files according to the constraints imposed by compilers.

2. (40 points) The two main requirements for the tools that belong to a PSE are that they be efficient and convenient. These requirements are achieved through *sharing, concurrency, interaction, and integration.* Explain (4 points) and illustrate (6 points) the meaning of these terms in this context (10 points for each term).

   Sketch of answer:

   (a) **Sharing:** various tools of the PSE share computer resources (memory, processor, devices, information) and thus allow their efficient usage. For example, the same copy of a C compiler may be shared by all programmers developing C programs.

   (b) **Concurrency:** various tools of the PSE may execute concurrently therefore increasing system performance. For example, a compiler may run in parallel with an editor, and in parallel with various components of the operating system servicing current program execution.

   (c) **Interaction:** tools in PSE interact with each other and with the system user making problem solving process convenient. This interaction is expressed as services tools can provide to each other and to the system user. For example, a compiler interacts with Operating System by asking the OS to perform I/O operations on its behalf. A compiler also interacts with its user by sending messages concerning syntax and semantic correctness of the programs it compiles. Shell interpreter interacts with the system user during problem solving process by receiving and executing user commands and sending messages to the user about the behavior of the problem solving process.

(d) **Integration:** tools of a PSE need to be integrated in the sense that if a tool T1 may provides a solution to the subproblems P1 of the problem solved by a tool T2 then T1 can be used as a component of T2. This contributes to both PSE tools convenience and efficiency.

3. (50 points) An assembler has been defined as a pair of mappings, $A = \langle HA, TA\rangle$, where $HA : AL_{Sem} \to ML_{Sem}$, $TA : AL_{Syn} \to ML_{Syn}$, which makes commutative the diagram in Figure **??**.

$$\begin{tikzcd}
AL_{Sem} \arrow[r, "Learn_{AL}"] \arrow[d, "\text{HA}"] & AL_{Syn} \arrow[r, "Eval_{AL}"] \arrow[d, "\text{TA}"] & AL_{Sem} \arrow[d, "\text{HA}"] \\
ML_{Sem} & ML_{Syn} \arrow[l, "Eval_{ML}"'] & ML_{Sem} \arrow[l, "Learn_{ML}"']
\end{tikzcd}$$

Figure 1: The assembler

You are required to answer the following questions:

(a) (10 points) The source language of the assembler is the assembly language $AL = \langle AL_{Sem}, AL_{Syn}, AL_{Sem} \leftrightarrow AL_{Syn}\rangle$. Explain the meaning of each of the components AL_{Sem}, AL_{Syn}, $AL_{Sem} \to AL_{Syn}$, $AL_{Syn} \to AL_{Sem}$ of the AL.

Solution sketch:

i. AL_{Sem} is the semantic of the assembly language and is defined as the computing system implementing machine computations.

ii. AL_{Syn} is the syntax of the assembly language and is defined by the collection of mnemonic notations programmers can use to represent machine computations.

iii. $AL_{Sem} \to AL_{Syn}$ is the collection of rules which allows programmers to express machine computations using mnemonic notations. These may be called *assembly language programming rules*

iv. $AL_{Syn} \to AL_{Sem}$ is the collection of rules which allows programmers to execute their assembly language programs thus simulating machine behavior.

(b) (10 points) The target language of the assembler is the machine language $ML = \langle ML_{Sem}, ML_{Syn}, ML_{Sem} \leftrightarrow ML_{Syn}\rangle$. Explain the meaning of each of the components ML_{Sem}, ML_{Syn}, $ML_{Sem} \to ML_{Syn}$, $ML_{Syn} \to ML_{Sem}$ of the ML (5 points). Explain the relationship between AL_{Sem} and ML_{Sem} (5 points).

Solution sketch:

i. ML_{Sem} is the semantic of the machine language and is defined as the computing system implementing machine computations.

ii. ML_{Syn} is the syntax of the machine language and is defined by the collection of machine codes programmers can use to represent machine computations.

iii. $ML_{Sem} \to ML_{Syn}$ is the collection of rules which allows programmers to express machine computations using the collection of codes recognized by the hardware. These may be called *machine language programming rules*

iv. $ML_{Syn} \to ML_{Sem}$ is the collection of rules which allows programmers to execute their machine language programs thus simulating machine behavior.

v. AL_{Sem} coincides with ML_{Sem}. Hence, the mappings $HA : AL_{Sem} \to ML_{Sem}$ is an identity map.

(c) (10 points) Describe the structure of the AL_{Syn} in terms of its three layers of generation.

Solution sketch: AL_{Syn} is defined on three levels of structuring:

i. On the first level are mnemonics denoting machine language operations, pseudo-operations, macro-operation definition and macro-operation call, general registers, index registers, modifiers, and address expressions denoting machine language operands.

ii. On the second level are assembly language statements. These are AL constructs of the form `[Label:]  Mnemonic [Operand] [Comment]` used to represent data and machine operations. The statement components in brackets are optional.

iii. On the third level are assembly language programs. These are defined as sequence of statements that start with a statement denoting program begin and end with a statement denoting program end.

(d) (10 points) Describe the structure of the ML_{Syn} in terms of its three layers of generation.

Solution sketch: ML_{Syn} is defined on three levels of structuring:

i. On the first level are the machined language codes. They are binary representations of opcodes, general registers, index registers, modifiers, and memory addresses.

ii. On the second level are machine language statements. These are specified by a fixed number of parameterized patterns used to represent data and operations.

iii. On the third level are machine language programs. These are defined as sequence of machine language instruction and data words assembled according to the model of program executions.

(e) (10 points) Sketch the implementation of the assembler component $TA : AL_{Syn} \to ML_{Syn}$ as a two pass assembler.

Solution sketch: The two pass assembler is implemented by two readings of the assembly language program called passes, which perform as follows:

Pass 1: read the assembly language program statement by statement. For each statement identify its generators and remember them and their machine language translations in appropriate tables. At the end of Pass 1 perform memory allocation and collect all external defined and referenced symbols in the External Symbol Dictionary, (ESD). **Note:** for optimization purpose the assembly language statements thus identified may be represented by appropriate patterns whose elements point to the generators and their translations stored in the appropriate tables. These patterns are written in the File of Internal Form (FIF), that save the time needed for the second reading of the assembly language source program.

Pass 2: is called by $Pass_1$ and performs the following tasks:

A. Traverse the symbol table and performs memory allocation.

B. Construct the External Symbol Dictionary, where imported and exported symbols used in the assembly language program are stored.

C. Read the assembly language program (or FIF if used) statement by statement. For each statement identify the machine language pattern representing the computation performed by the statement and instantiate it using the translations of the generators stored in tables by Pass 1.

D. Writes thus instantiated pattern into the File of Object Generated (FOG) generated by the assembler. i

E. Collects the address dependent constants into the Relocation and Linking Directory and generate the Machine Object Module, $MOM = \langle ESD, Text, RLD\rangle$ where $Text$ is the program generated in FOG.

Problem III: reliability and efficiency of BOS (100 points)

The action Solve below represents the evolution of the control program to a reliable and efficient batch operating system, called *Multiprogramming System*.

```
Solve::
    local $JobFile$: file of Jobs,
        $Job1, Job2, Job3$: job data structures;
    repeat
        $l_1$:$Batch(Job1, JobFile)$ or $Skip$ :$\hat{l}_1$
          ||
        $l_2$:$Process(Job2, JobFile)$ or $Skip$ :$\hat{l}_2$
          ||
        $l_3$: $DoIO(Job3, JobFile)$ or $Skip$ :$\hat{l}_3$
    forever
```

You are required to answer the following questions regarding the Solve action:

1. (30 points) What are the hardware (10 points) and software (20 points) mechanisms used to make the control program reliable and efficient thus evolving it to the Solve action?

   Solution sketch: Hardware mechanisms: (1) Mode Bit (MB), Interrupt mechanism, Protection Memory (PM), Protection Key (PK) in PSW, used to ensure reliability; (2) Disk systems where JDS can be accessed directly, used to ensure efficiency by overlapping processor, I/Devices, and users, operations.

   Software mechanisms: system call (supervisor call), table of functions performing services, a dispatcher program that is initiated when a supervisor call is issued by a user program, and Job Summary Table (JST) that summarize the JDS. JST is organized as a linked list and is maintained in main memory. JobFile is split into three queues:

   (a) InList, which holds the new jobs arrived in the system, InList is manipulated by a long-term scheduling algorithm usually called InSpooler;

   (b) OutList, which hold the jobs that are ready to depart from the system; OutList is manipulated by a long-term scheduling algorithms usually called an OutSpooler;

   (c) ReadyList which hold the jobs that are currently competing for processor execution. Ready list is manipulated by a short-term scheduler algorithm.

2. (20 points) Explain the communication protocol enforced by the hardware and software mechanisms that makes Solve reliable and efficient.

   Solution sketch: Communication protocol: System programs run in supervisor mode (MB =1) while programs initiated by the control program run in user mode

(MB=0). MB splits the instructions performed by the processor in two disjoint classes: privileged instruction (PI) and non-privileged instructions (NP). If an instructions $i \in NP$ is initiated while $MB = 1$ an interrupt is generated by the processor and the control is transferred to the control-program. This allows the system programs access to the entire repertoire of instructions executed by the processor while the user programs can perform only non-privileged instructions. A special privileged instruction called supervisory call, SC, is provided which allows the user program to ask the control program to execute a function on its behalf. Thus, a user program can ask the CP to perform a service on his behalf issuing a supervisor call which takes as parameter the service requested. Similarly, PK and PM allow the system to control the memory access, thus protecting the programs that share the memory against each other unauthorized actions.

3. (50 points) The performance of the system was defined as the ratio $\frac{ProcessorTime}{TotalTime}$ where $ProcessorTime$ is the time used effectively by the processor performing user computations and $TotalTime$ is the total time used by the computer to solve a problem.

   (a) Explain the concept of *overlapping processing* performed by BOS (10 points). How is the system performance affected by overlapping processing? (10 points).

   Solution sketch: Overlapping processing: is the processing mode performed by BOS where I/O operations of a program are performed by I/O devices in parallel with the processing operations of the same program performed by the control processor. The system performance can be improved by buffering. The performance improvement is however limited by the difference in speed between I/O devices and processor.

   (b) Explain the concept of *multiprogramming system* that implements the action Solve (10 points). How is the system performance affected by multiprogramming processing? (10 points).

   Solution sketch: Multiprogramming system: is the processing mode performed by Solve where I/O operations of a program can be overlapped with processing operations of another program. This is achieved by using disk systems and maintaining the JobFile into three lists as explained above. The performance of multiprogramming system can be increased with the number of programs maintained in the ReadyList which is called multiprogramming degree.

   (c) Explain the concept of the *thrashing state* of a multiprogramming system (10 points).

   Solution sketch: Thrashing state: when the multiprogramming degree increases above certain number, called threshold, the system performance decreases dramatically. The processor become busy switching jobs among the queues existent in the system rather than performing user computations. The state of the system where performance start decreasing as number of programs in the

ReadyList increases is called *thrashing state*. To maintain system performance high the multiprogramming degree needs to be kept below the threshold.