Department of Electrical Engineering and Computer Science

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.033 Computer Systems Engineering: Spring 2004

Quiz I

All problems on this quiz are multiple-choice questions. In order to receive credit you must fill in the blank(s) or mark the correct answer or answers for each question. You have 50 minutes to answer this quiz.

Write your name on this cover sheet AND at the bottom of each page of this booklet.

Some questions may be much harder than others. Read them all through first and attack them in the order that allows you to make the most progress. If you find a question ambiguous, be sure to write down any assumptions you make. Be neat. If we can't understand your answer, we can't give you credit!

THIS IS AN OPEN BOOK, OPEN NOTES QUIZ.
NO PHONES, NO LAPTOP, NO PDAS, ETC.

CIRCLE your recitation section number:

10:00	1. Ernst/Strauss	2. Madden/Hickey	
11:00	3. Dabek/Hickey	4. Ernst/Chen	5. Madden/Strauss
12:00	6. Dabek/Chen		
1:00	7. Katabi/Bicket	8. Saltzer/Garfinkel	9. Karger/Lesniewski
2:00	10. Saltzer/Bicket	11. Katabi/Lesniewski	12. Karger/Garfinkel

Do not write in the boxes below

1-3 (xx/30)	4-5 (xx/20)	6-10 (xx/50)	Total (xx/100)

Name:

I Reading questions

1. [10 points]: Which of the following is an example of a primitive designed for sequence coordination in UNIX (reading #6, "The UNIX time-sharing system")?

(Circle ALL that apply)

A. wait

B. read on a pipe

C. lseek

D. mount

2. [12 points]: The Flash paper ("Flash: An efficient and portable Web server," reading # 7), describes the architectures: SPED, AMPED, MP, and MT. For each of the following statements, circle the architectures that apply to the statement:

(Circle ALL that apply)

A. Suppose a single client issues requests serially (i.e., starting the next request after the response on the current request has been received) for files located on the disk. Which of the following architectures provides a significant improvement in throughput over the SPED architecture.
Choose from: MP, MT, AMPED, None, All.

B. Provides the best performance when the cache miss rate is high when serving multiple clients.
Choose from: SPED, MP.

C. Makes use multiple times of the UNIX fork supervisor call.
Choose from: SPED, APMED, MT, MP.

D. Assume that the delay observed for a client for a web request for a cached page is broken down as follows: 100 ms network latency to deliver the request packet to the server, 1ms of processing time at the server, 100 ms of network latency to deliver the page to the client in a return packet. A number of clients issue requests at the same time. Which of the following architectures can process more than one request every 201ms.
Choose from: MP, MT, SPED, AMPED, None, All.

3. [8 points]: If eight stations on an Ethernet ("Ethernet: distributed packet switching for local computer networks", reading #8) all want to transmit one packet, which of the following statements is true?

(Circle best answer)

A. It is guaranteed that all transmissions will succeed.

B. It is most likely (i.e., with high probability) that all stations will eventually end up being able to transmit their data successfully.

C. Some of the transmissions may eventually succeed, but it is likely some may not.

D. It is likely that none of the transmissions will eventually succeed.

II Local remote procedure call

Ben Bitdiddle is hired to enforce modularity in a large banking application. He splits the program into two pieces: the client and the server. Ben wants to use remote procedure calls to communicate between the client and server, which both run on the same physical machine with one processor. So, he sets out to design a *local remote procedure call (LRPC)* system using the kernel interface from the class notes. (You can find this interface in Table 2-1 in Section 2.D.11, but you should be able to make it through the question without having to consult the notes.).

The kernel interface is implemented by the kernel program, which runs in the kernel address space. The kernel manages the address spaces and threads. The client and the server each run in their own user address space. The client and server each start with one thread. There are no user programs other than the client and server running on the machine.

Communicating messages. To implement LRPC, the client and server share a block of memory through which they communicate requests and responses. This block is at physical address *WELLKNOWNBLOCK* and contains a *request* and a *response*, each of the type *structure message*[1]:

```
structure message {
   char data[1024];  // the data of the marshalled message (at most 1024 bytes)
   int size;         // the size of the data
   bool present;     // is a message present?
};

structure message request;  // request is an instance of structure message
structure message response;  // response is an instance of structure message
```

The client and server both map the block of memory at physical address *WELLKNOWNBLOCK* into their address spaces. The client maps it at the virtual address *CLIENT* and the server maps it at the virtual address *SERVER*.

Server operation. To prepare for receiving an LPRC, the server registers with the kernel a virtual address (*STACK*) to be used as stack pointer and an entry point (the virtual address *receive*, where the procedure RECEIVE is located). The server maps the well-known block in its address space *server_asn* (the identifier for the server address space). Then, it calls the supervisor call YIELD so that YIELD can schedule another thread, which might perform an LRPC to the server.

procedure SERVER()
 $register\_gate$(*STACK*, $receive$); *// register entry point; see below*
 $allocate\_block$(*WELLKNOWNBLOCK*); *// allocate block of physical memory*
 map($server\_asn$, *WELLKNOWNBLOCK*, *SERVER*); *// map it at SERVER*
 while ($true$) **do** $yield$();

[1] Read the code in this section II carefully; it will take time, but you should have enough time.

Name:

When a client thread performs an LRPC, the kernel starts the LRPC at address *receive* in the server's address space, invoking the procedure RECEIVE, and running with the server's stack at virtual address *STACK*:

procedure RECEIVE()
 do_RPC(*SERVER*.$request.data$, *SERVER*.$request.size$, *SERVER*.$response$);
 SERVER.$response.present \leftarrow true$;

The notation *SERVER.request.data* means the named data field (*data*) of the named structure (*request*) located on the page with the named virtual address (*SERVER*).

The procedure RECEIVE invokes the procedure call DO_RPC, which takes the request message as an argument, unmarshals the message, processes it, and when it returns it has filled in the response message with the marshalled result. After DO_RPC returns, the procedure RECEIVE sets *SERVER.response.present* to true, indicating to the client that the response is available. The exact way in which RECEIVE returns to the client is not specified in this problem but you may assume it works correctly.

4. [10 points]: In which address space is the code that implements the supervisor call YIELD located (i.e., the code that schedules the processor)?

(Circle best answer)

A. The kernel address space

B. The client address space

C. The server address space

D. In all three.

Client operation. At initialization time, the client maps the block at physical address *WELLKNOWNBLOCK* to virtual address *CLIENT* in its address space *client_asn*:

procedure INIT_CLIENT()
 map(*client_asn*, *WELLKNOWNBLOCK*, *CLIENT*);

To perform an LRPC to the server, the client calls LRPC with the server's address space number *server_asn*, a request buffer (containing the marshalled request) and the request size. LRPC returns a response buffer (containing the marshalled result) and response size:

Name:

procedure LRPC($int\ asn, char\ request[], int\ request\_size, char\ response[], int\ response\_size$)
 $copy(request, CLIENT.request.data, request\_size)$;
 $CLIENT.request.size \leftarrow request\_size$;
 $CLIENT.request.present \leftarrow true$;
 $CLIENT.response.present \leftarrow false$;
 $transfer\_to\_gate(asn, receive)$;
 while ($CLIENT.response.present \neq true$) **do** {
  $yield()$;
 }
 $copy(CLIENT.response.data, response, CLIENT.response.size)$;
 $response\_size \leftarrow CLIENT.response.size$;

The client copies the bytes from the request buffer into the part of its address space corresponding to *WELLKNOWNBLOCK*. (The procedure COPY(SRC, DST, LEN) is a procedure in the client's address space that copies len bytes from *src* to *dst*). Then, it calls TRANSFER_TO_GATE to transfer control to the code at the address *receive* in the server's address space. If the server hasn't generated a response, it calls the supervisor call YIELD to yield the processor. Once the response message is available, the client returns the result (*response* and *response_size*) to the caller of LRPC.

5. [10 points]: For Ben's LPRC to work correctly must the virtual addresses *SERVER* and *CLIENT* have the same value?

(Circle best answer)

- **A.** No, as long as the addresses *SERVER* and *CLIENT* in both address spaces translate to the same physical block *WELLKNOWNBLOCK*.
- **B.** No, the virtual addresses can map to any physical address.
- **C.** Yes, because otherwise *SERVER* and *CLIENT* will translate to different physical addresses.
- **D.** Yes, because otherwise there is no fault isolation.

Implementing gates. Since Ben cannot find the code for the supervisor calls REGISTER_GATE and TRANSFER_TO_GATE in the class notes, he implements them as follows. First, the kernel maintains a structure for each address space in the array *entrypoint*:

```
structure entry {
   int stack;  // value of stack pointer
   int entry_addr; // virtual address of entry point
   int pmar; // page map address register
} entrypoint[10];   // this kernel manages at most 10 address space
```

In this structure the kernel stores for each address space the address of the stack (*stack*) to be used to invoke a procedure at the address *entry_addr*. The structure also contains the *pmar* for each address space. Except for the kernel address space (number 0), all *pmar* entries in *entrypoint* have the user-mode bit switched on and interrupt-enable bit switched on.

Name:

The supervisor call REGISTER_GATE registers with the kernel the virtual address of the stack to be used for calls to the procedure stored at virtual address *addr*:

procedure REGISTER_GATE($stack, addr$)
 $entrypoint[current\_asn].stack \leftarrow stack$;
 $entrypoint[current\_asn].entry\_addr \leftarrow addr$;
 $entrypoint[current\_asn].pmar \leftarrow current\_pmar$;

The variable *current_asn* and *current_pmar* refer to the user address space and its page map address register that made this supervisor call. In the case of the server, *current_asn* is equal to *server_asn* and *current_pmar* is equal to the server's pmar.

The supervisor call TRANSFER_TO_GATE transfers the processor to the virtual address *addr* in address space *asn*. The supervisor call first checks if *addr* is an entry point that is registered with the kernel, and if so, it pushes the address on the stack registered with the kernel:

procedure TRANSFER_TO_GATE($asn, addr$)
 if $(entrypoint[asn].entry\_addr = addr)$ **then** {
 $R0 \leftarrow entrypoint[asn].pmar$;
 $R1 \leftarrow user\_to\_kernel(asn, entrypoint[asn].stack)$; *// translate stack into kernel's address space*
 $(R1) \leftarrow addr$; *// push addr (4 bytes) on stack in user address space*
 $SP \leftarrow entrypoint[asn].stack$;
 $SP \leftarrow SP - 4$; *// lower stack pointer 4 bytes*
 jmp(*LEAVE*);
 }
 else $return$(*ERROR*);

You may assume that registers R0 and R1 are available for use by the kernel.

Then, TRANSFER_TO_GATE jumps to the *LEAVE* stub (copied from the class notes):

```
LEAVE:
  MOVE R0, PMAR   // move the content of R0 into the PMAR register,
                  // which may result in an addres space switch
  RTE             // pop return address and load it into the PC register
```

As described on page 2-47 of the notes, *LEAVE* sets the *pmar* of the processor to the page table of the address space *asn*. As described in the notes, *LEAVE* is mapped in each address space at the same virtual address.

Name:

6. [10 points]: During the execution of the procedure RECEIVE how many threads are running or are in a call to YIELD in the server address space?

(Circle best answer)

A. 0

B. 1

C. 2

D. 2 or more

7. [10 points]: How many supervisor calls could the client perform in the procedure LRPC?

(Circle best answer)

A. 1

B. 2

C. 3

D. 4

E. 2 or more

8. [10 points]: Ben's goal is to enforce modularity. Which of the following statements are true statements about Ben's LRPC implementation?

(Circle ALL that apply)

A. The client thread cannot transfer control to any address in the server address space

B. The client thread cannot overwrite any physical memory that is mapped in the server's address space.

C. After the client has invoked TRANSFER_TO_GATE in LRPC, the server is guaranteed to set *SERVER.response.present* to true at some point.

D. The procedure LPRC ought to be modified to check the response message and process only valid responses.

Name:

9. [10 points]: Assume that REGISTER_GATE and TRANSFER_TO_GATE are also used by other programs. Which of the following statements is true about Ben's implementation of REGISTER_GATE and TRANSFER_TO_GATE?

(Circle ALL that apply)

A. The kernel might use an invalid address when writing the value *addr* on the stack passed in by a user program.

B. A user program might use an invalid address when executing the *RTE* instruction in *LEAVE*.

C. The kernel transfers control to the server address space with the user-mode bit switched off.

D. The kernel enters the server address space only at the registered address *entry_addr*.

Ben modifies the client to have multiple threads of execution. If one client thread calls the server and DO_RPC calls YIELD, another client thread can run on the processor.

10. [10 points]: Which of the following statements is true about Ben's implementation of LRPC with multiple threads?

(Circle ALL that apply)

A. On a single-processor machine, there can be race conditions when multiple clients threads call LRPC, even if the kernel schedules the threads nonpreemptively.

B. On a single-processor machine, there can race conditions when multiple clients threads call LRPC and the kernel schedules the threads preemptively.

C. On multiprocessor computer, there can be race conditions when multiple clients threads call LRPC.

D. It is impossible to have multiple threads if the computer doesn't have multiple physical processors.

End of Quiz I

Name: