```
void foo (char* x, int n) {
  char* y = copy (x, n);
  printf ("%s", y);
}
```

Exercise 6

Write the interrupt driven version of the following code that prints the sum of the first 256 bytes of disk block 1234. Assume that async_read correctly performs the PIOs to request the block data be transferred into buf and that it enques sufficient information so that the interrupt service routine can call the specified event handler.

```
char buf[256];
void ps() {
  async_read (1234, buf, 256);
  int s=0;
  for (int i=0; i<256; i++)
    s += buf[i];
  printf ("%d\n", s);
}
```

10 (6 marks) **Using Function Pointers.** Write a C procedure named `apply` that returns either the minimum or maximum value of a list of non-negative integers, as determined by one of its parameters, a function pointer. If the list is empty it should return -1. Assume the existence of procedures named `min(a,b)` and `max(a,b)` that compare two integers and returns the min or max integer. No other procedures are allowed and `apply` is not permitted to use an `if` statement. For example, the following statement should compute the maximum value in list `a` of length `n`.

```
int m = apply (max, a, n);
```

Write the procedure `apply()`:

11 (6 marks) **IO Devices.** For each of the following, (a) explain what it is and (b) state whether the CPU or IO controller determines when it occurs (i.e., initiates it).

11a Programmed IO (PIO):

11b DMA:

11c Interrupts:

9 **(6 marks)** **Switch Statements.** There are two ways to implement `switch` statements in machine code. For purposes of this question, lets call them A and B.

9a Describe A, very briefly.

9b Describe B, very briefly.

9c State precisely one situation where A would be preferred over B and why.

9d State precisely one situation where B would be preferred over A and why.

10 **(9 marks)** **IO Devices.** Three key hardware features used to incorporate IO Devices with the CPU and memory are Programmed IO (PIO), Direct Memory Access (DMA) and interrupts.

10a Carefully explain the difference between PIO and DMA; give one advantage of DMA.

10b Demonstrate why interrupts are needed by carefully explaining what programs would have to do differently to perform IO if interrupts didn't exist and what disadvantages this approach would have.

10c Explain how interrupts would be added to the Simple Machine simulator by indicating where the interrupt-handling logic would be added and saying roughly what it would do.

Exercise 4

It prints 120.

Exercise 5

Yes, a memory leak is possible. It can be fixed as follows:

```
char* copy (char* from, int n) {
  char* to = malloc (n);
  for (int i=0; i<n; i++)
    to[i] = from[i];
  return to;
}
void foo (char* x, int n) {
  char* y = copy (x, n);
  printf ("%s", y);
  free (y);
}
```

Exercise 6

```
void getsum (char* buf, int n) {
  int s=0;
  for (int i=0; i<256; i++)
    s += buf[i];
  printf ("%d\n", s);
}

char buf[256];

void ps() {
  int s = async_read (1234, buf, 256, getsum);
}
```

```
int  c = 0;
int* v[2];

void bar (int* ap) {
  if (v[c] == NULL || *ap < *v[c]) {
    if (v[c] != NULL)
      dec(v[c]);
    v[c] = ap;
    inc(v[c]);
  }
  c = (c + 1) % 2;
}

void zot (int a) {
  if (v[c] != NULL && a < *v[c])
    *v[c] = a;
  c = (c + 1) % 2;
}
```

10 **(6 marks)** **Using Function Pointers.** Write a C procedure named `apply` that returns either the minimum or maximum value of a list of non-negative integers, as determined by one of its parameters, a function pointer. If the list is empty it should return -1. Assume the existence of procedures named `min(a,b)` and `max(a,b)` that compare two integers and returns the min or max integer. No other procedures are allowed and `apply` is not permitted to use an `if` statement. For example, the following statement should compute the maximum value in list `a` of length `n`.

```
int m = apply (max, a, n);
```

Write the procedure `apply()`:

```
int apply (int (*f)(int, int), int* a, int n) {
    if (n<=0)
        return -1;   // for 2016W2 this isn't necessary.
    else {
        int v = a[0];
        for (int i=1; i<n; i++)
            v = f(v, a[i]);
        return v;
    }
}
```

11 **(6 marks)** **IO Devices.** For each of the following, (a) explain what it is and (b) state whether the CPU or IO controller determines when it occurs (i.e., initiates it).

11a Programmed IO (PIO):

11b DMA:

11c Interrupts:

12 **(8 marks)** **Threads and Scheduling.** Answer the following questions about threads.

12a Explain briefly (without giving any code) how threads can be used to simplify code that performs asynchronous operations such as communicating with an IO controller.

12b Explain briefly the difference between `uthread_yield` and `uthread_block`.

12c Is it possible for a thread to unblock itself? Explain your answer.

12d Consider a system in which there is at least one thread on the *ready queue* when a thread unblocks. What happens to the unblocked thread? Explain.

13 **(9 marks)** **Synchronization** Consider the following C code that uses mutexes and condition variables. The code is considered to be is correct if both procedures complete successfully when they are called from concurrent threads

9 **(6 marks)** **Switch Statements.** There are two ways to implement `switch` statements in machine code. For purposes of this question, lets call them A and B.

9a Describe A, very briefly.

A sequence of `if` statements.

9b Describe B, very briefly.

A jump table of labels corresponding to switch-cases.

9c State precisely one situation where A would be preferred over B and why.

If there are very few cases to consider, then the overhead of using a jumptable is higher than a few statements. Reading memory is much slower than executing a conditional branch.

OR: If the case values are spread apart (sparsely populated), there will be a lot of wasted memory in the jumptable because we have to represent a contiguous range of values in a jump table. e.g.

```
switch (i) {
  case 1:
    j = 5;
    break;
  case 1000000:
    j = 10;
    break;
 }
```

would require a jump table with one million elements!

9d State precisely one situation where B would be preferred over A and why.

When you have lots of cases to check and their values are close together (densely populated), the jump table is the best choice. When there are N cases, it takes `O(N)` to test all cases, whereas it will always take `O(1)` with a jump table. If the values are closer together, then we waste less memory creating the jump table.

10 **(9 marks)** **IO Devices.** Three key hardware features used to incorporate IO Devices with the CPU and memory are Programmed IO (PIO), Direct Memory Access (DMA) and interrupts.

10a Carefully explain the difference between PIO and DMA; give one advantage of DMA.

The CPU uses PIO to read or write to an IO device one word at a time. IO Devices use DMA to read or write memory directly, without involving the CPU. An advantage of PIO is that the CPU can use it to transfer data to an IO device, or control it; e.g., to signal the IO device that the CPU wants something, Another advantage is that PIO has lower overhead and lower latency for very small transfers because it avoids the overhead of setting up a DMA. The advantage of DMA is that the transfer occurs asynchronously to the CPU and so the CPU is free to do other things during the transfer. For any transfer larger than 64-128 bytes, DMA typically transfers with lower overhead and latency than PIO.

10b Demonstrate why interrupts are needed by carefully explaining what programs would have to do differently to perform IO if interrupts didn't exist and what disadvantages this approach would have.

If interrupts didn't exist, a program would have to repeatedly *poll* the IO device to determine whether the device had information (e.g., keyboard presses) for it. The disadvantages of polling are that it wastes CPU cycles unnecessarily when the IO device doesn't have information for the CPU. If polling is very frequent, then this overhead is very high.

Infrequent polling may not be a suitable solution either because it increases the latency (i.e., delay) between when an IO device notifies the CPU that it wants its attention, and when the CPU actually notices it. This would increase, for example, the latency of disk and network reads. There is an undesirable tradeoff between latency and CPU 'wastage'.

10c Explain how interrupts would be added to the Simple Machine simulator by indicating where the interrupt-handling logic would be added and saying roughly what it would do.

Insert a new stage just before the fetch stage (or after the execute stage). Check there to see if there is an interrupt pending and if so, jump to the address specified in the interrupt vector table. This will invoke the specific interrupt service routine for the given type of interrupt.

11 (10 marks) Threads.

11a Threads can be used to manage the asynchrony inherent in I/O-device access (e.g., reading from disk). Carefully explain how threads help.

Because threads execute asynchronously themselves, you can write code that looks synchronous but executes asynchronously. This is easier to read, write, and think about for programmers.

IO can take quite long (eg millions of CPU cycles), and in this time the CPU can be doing other things. Threads provide this flexibility because they can easily be blocked when they are waiting for something, and the CPU can go ahead and execute other threads.

11b Carefully describe in plain English the sequence of steps a user-level thread system such as *uthreads* follows to switch from one thread to another. Ensure that your answer explains the role of the *ready queue* and explains how the hardware switches from running one thread to the other.

A thread switch occurs in uthreads when a thread calls `uthread_block()` or `uthread_yield()`. This places the current thread on the ready queue and dequeues something else off the ready queue to begin execution.

The actual transfer of execution is achieved by pushing the register of the first thread onto its stack, saving its current stack pointer in its TCB and then switching the value of the stack pointer register to point to the target thread's stack. Then the CPU pops the registers of this new thread from its stack, and resumes execution.

11c What is the role of the *thread scheduler*?

The thread scheduler decides which thread should execute next.

11d Explain priority-based, round-robin scheduling.

Each thread has an assigned priority. When a thread finishes execution (or blocks/yields), the next thread chosen to execute is the thread with the highest priority.

11e Explain what else is needed to ensure that threads of equal priority get an equal share of the CPU?

In the above explanation of priority-based, round-robin scheduling *starvation* is possible because a single high priority thread could dominate all the CPU time if it never finishes execution. *Quantum preemption* would allow for all threads of equal priority to get an equal share of the CPU by periodically interrupting and switching threads.

12 (16 marks) Synchronization

12a Explain the difference between busy-waiting and blocking. Give one advantage of blocking.

Busy waiting occurs when the a thread waits for a lock to be free by actively polling the lock's value, spinning in a loop repeatedly reading it until it sees that it is available.

Blocking waiting occurs when a thread waits for a lock by sleeping so that other threads can use the CPU. It is then the responsibility of the thread that releases the lock to wakeup the waiting thread.