Name: ______________________________

Write your name and computing ID above. Write your computing ID at the top of each page in case pages get separated. Sign the honor pledge below.

Generally, we will not answer questions about the exam during the exam time. If you think a question is unclear and requires additional information to answer, please explain how in your answer. For multiple choice questions, write a ⋆ next to the relevant option(s) along with your explanation.

On my honor as a student I have neither given nor received aid on this exam.

____

1. Suppose the following sequence of events happens on a single-core Unix-like system running three single-threaded processes A, B, and C:

   1. Process A opens a file
   2. Process A starts to read from the file, but the data must first be transferred from the disk to memory
   3. While process A is waiting, process B runs
   4. Process B performs some computations
   5. Process B is paused temporarily, and Process C runs and performs some computations
   6. Process B resumes running and sends a signal to process C
   7. Process C's signal handler starts running
   8. Process C's signal handler prints out a message to the terminal
   9. The contents of process A's file are retrieved from disk, and as a result Process A resumes running

(a) (4 points) During which of the steps above is a system call likely to occur? (Write step numbers from the list above.)

Solution: 1268

(b) (4 points) During which of the steps above is a non-system-call exception likely to occur? (Write step numbers from the list above.)

Solution: 569

(c) (4 points) While process C's signal handler starts running, the value of the register `%r8` in Process A is most likely stored ____________________.

- □ on the disk's controller
- □ in one of the registers on the processor
- □ on process C's stack
- □ on process B's stack
- ■ in some part of the operating system's memory
- □ none of the above, explain: ____________________

(d) (4 points) Immediately after process C's signal handler starts, ____________________. **Select all that apply.**

- □ the processor will be in kernel mode
- □ process C's registers or stack will contain a pointer to process B's code that triggered the signal
- □ process C's registers will have a pointer to process B's stack
- ■ local variables used during process C's computations (step 5) will be on process C's stack

2. For the following questions:

- consider a two-way 2^{17} byte cache with 2^8-byte blocks, an LRU replacement policy, a write-allocate policy, and a write-back policy, and
- assume all addresses are physical addresses and that virtual memory is not in use

(a) (4 points) The value at the address `0x123456` will be stored in the same set of the cache as the value from address ________________. **Select all that apply.**

□ `0x123500`

■ `0xF23456`

□ `0x123123`

□ `0x12345`

(b) (12 points) Select the correct options to complete the following table.

For the 'write to main memory while handling?' column, select 'yes' if before completing the cache read or write, the cache should start a write to the main memory (or the next level of cache).

Assume the cache is initially empty and all accesses are to a single byte, and performed in the order shown below. *-1 per wrong*

read/write	tag	set index	offset	hit/miss?	write to memory while handling?
write	0x10	0	4	□ hit ■ **miss**	□ yes ■ no
write	0x20	0	8	□ hit ■ **miss**	□ yes ■ no
write	0x40	0	0	□ hit ■ miss	■ yes □ no
read	0x20	0	4	■ hit □ miss	□ yes ■ no
read	0x3F	0	0	□ hit ■ miss	■ yes □ no
read	0x20	0	8	■ hit □ miss	□ yes ■ no
read	0x10	0	4	□ hit ■ miss	□ yes ■ no

(c) (8 points) Consider the following C snippet:

```
array2[array1[x] * 0x1000]
```

where `array2` and `array1` are both arrays of 1-byte unsigned chars. Assume the compiler does not optimize away the accesses to the two arrays (even if those values are not used).

Suppose array1 and array2 both have physical addresses which are multiples of 2^{20} (and contiguous in physical memory). We determine (via a PRIME+PROBE side channel) that executing the above C snippet evicts from cache sets with index `0x10` and `0x20`. Based on this information, give a possible value of `x` and of `array1[x]`:

- `x`: 0x1000 to 0x10FF or 0x2000 to 0x20FF or these + multiples of 0x10000
- `array1[x]`: 1 or 2

Use the box below to show any work:

Solution:

3. This set of questions refers to the following assembly function (which is also reproduced on the next page):

```
countOnes:
        xorl    %ecx, %ecx
        xorl    %eax, %eax
.L2:
        movl    %edi, %edx        /* A */
        shrl    %cl, %edx         /* B */
        incl    %ecx              /* C */
        andl    $1, %edx          /* D */
        addl    %edx, %eax        /* E */
        cmpl    $32, %ecx         /* F */
        jne     .L2               /* G */
        ret
```

(a) Suppose the `countOnes` function above is executed on a five-stage pipelined processor (where the pipeline stages are fetch, decode (register read), execute, memory, writeback).

i. (5 points) During one iteration of the loop that starts at `.L2` and ends at `jne .L2`, we would expect which of the following forwarding to occur? (It's possible that not all needed forwarding is listed.) **Select all that apply.**

■ of `%edx`'s value from A to B

□ of `%edx`'s value from A to D

□ of `%ecx`/`%cl`'s value from B to C

■ of `%edx`'s value from B to D

■ of `%edx`'s value from D to E

ii. (4 points) If the processor predicts the `jne .L2` as taken, then, during the second iteration of the loop, we would expect instruction A's decode stage to run at the same time as an instance of instruction ________'s memory stage.

□ B (`shrl %cl, %edx`)

□ C (`incl %ecx`)

□ D (`andl $1, %edx`)

□ E (`addl %edx, %eax`)

■ F (`cmpl $32, %ecx`)

□ G (`jne .L2`)

□ none of the above

```
countOnes:
        xorl    %ecx, %ecx
        xorl    %eax, %eax
.L2:
        movl    %edi, %edx        /* A */
        shrl    %cl, %edx         /* B */
        incl    %ecx              /* C */
        andl    $1, %edx          /* D */
        addl    %edx, %eax        /* E */
        cmpl    $32, %ecx         /* F */
        jne     .L2               /* G */
        ret
```

(b) For the following questions, consider the `countOnes` function executing on an out-of-order processor designs that can fetch, rename, execute, and writeback multiple instructions per cycle. Answer the following questions about what would be possible to be done simultaneously assuming enough capacity (in terms of instruction queue size, reorder buffer size, number of instructions fetched per cycle, etc.)

i. (5 points) After register renaming, it is possible for two instances of instruction C to be in the instruction queue at the same time. What could be true about these two instances of instruction C? **Select all that apply.**

- □ they could have the same destination register
- ■ they could have different destination registers
- □ they could have the same source register
- ■ they could have different source registers
- ■ the destination register of one could be the same as the source register of the other

ii. (5 points) Within one iteration of the loop above, what instructions could compute their results at the same time as instruction D (`andl $1, %edx`) computes its results. **Select all that apply.**

- □ instruction A (`movl %edi, %edx`)
- □ instruction B (`shrl %cl, %edx`)
- ■ instruction C (`incl %ecx`)
- □ instruction E (`addl %edx, %eax`)
- ■ instruction F (`cmpl $32, %ecx`)

iii. (4 points) It is possible for the processor to compute the results of instruction ________ from two different iterations of the loop at the same time. **Select all that apply.**

- ■ instruction B (`shrl %cl, %edx`)
- □ instruction C (`incl %ecx`)
- ■ instruction D (`andl $1, %edx`)
- □ instruction E (`addl %edx, %eax`)

4. (16 points) Consider a system for reserving tickets to some event. The system is implemented using multiple threads, where each customer has a dedicated thread to perform actions on their behalf. Tickets can either be *available*, *pending*, or *purchased*.

   When a customer purchases a ticket, their thread uses the `ReserveTicket` function to mark it as pending. This will wait for a ticket to be available or for all tickets to be purchased. If all tickets are purchased, the function returns NULL; otherwise, the customer either decides to purchase it, using the `PurchaseTicket` function, or declines to purchase it using the `DeclineToPurchaseTicket` function.

   Fill in the THREE blanks in the following C code for implementing this scheme using monitors. (You may find it helpful to refer to the reference sheet's list of POSIX functions.)

```
typedef struct {
    int is_pending;
    int is_available;
    int is_purchased;
    const char *other_info;
} Ticket;

Ticket tickets[NUM_TICKETS];
pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;
int available_tickets = NUM_TICKETS;
int pending_tickets = 0;
int purchased_tickets = 0;
pthread_cond_t available_cv = PTHREAD_COND_INITIALIZER;

/* Wait for a ticket to be available or for all tickets to be purchased.
 *
 * Return NULL if all tickets are purchased; Otherwise reserve a ticket (marking it
 * and return it.
 */
Ticket *ReserveTicket() {
    pthread_mutex_lock(&lock);

    while (/* BLANK 1: */ available_tickets == 0 &&
                          purchased_tickets != NUM_TICKETS) {
        pthread_cond_wait(&available_cv, &lock);
    }
    Ticket *result = NULL;
    if (purchased_tickets != NUM_TICKETS) {
        available_tickets -= 1;
        pending_tickets += 1; /* WAS MISSING ON EXAM */
        for (int i = 0; i < NUM_TICKETS; ++i) {
            if (tickets[i]->is_available) {
                tickets[i]->is_available = 0;
                tickets[i]->is_pending = 1;
                result = tickets[i];
                break;
            }
        }
    }
    pthread_mutex_unlock(&lock);
    return result;
}

void PurchaseTicket(Ticket *ticket) {
    pthread_mutex_lock(&lock);
    ticket->is_pending = 0;
    ticket->is_purchased = 1;
    purchased_tickets += 1;
    pending_tickets -= 1;
    if (purchased_tickets == NUM_TICKETS) {
        /* BLANK 2: */ pthread_cond_broadcast(&available_cv);
    }
    pthread_mutex_unlock(&lock);
}

void DeclineToPurchaseTicket(Ticket *ticket) {
    pthread_mutex_lock(&lock);
    ticket->is_pending = 0;
    ticket->is_available = 1;
    available_tickets += 1; pending_tickets -= 1;

    /* no deduction for using broadcast here */
    /* BLANK 3: */ pthread_cond_signal(&available_cv);
    pthread_mutex_unlock(&lock);
}
```

5. Suppose two programs A and B communicate over a network using UDP. (Recall that UDP provides the 'mailbox' model of communication we discussed in lecture, where each message is called a 'datagram'.)

Program B accepts the following commands:

- to list the files in a directory with a particular name; program B will send the list of files in reply
- to create a file with a particular name and particular contents; program B will send a boolean indicating whether the creation was successful in reply

Program B receives each command in a single datagram, and sends any reply as as a single datagram.

For the following questions, assume that datagrams may be **lost** or **reordered**, but that **datagrams are not otherwise corrupted in transit**.

(a) (5 points) Suppose A and B are on different local networks, so their UDP datagrams need to be sent via multiple intermediate routers.

Refer to the reference sheet's summary of layers in the TCP/IP networking model.

When program B sends a reply to program A, the machine on which **program B** is running will most likely send a frame which will ________________________. **Select all that apply.**

□ have a destination MAC address of the router closest to A

■ have a destination MAC address of the router closest to B

□ contain a packet with a destination IP address identifying the router closet to A

■ contain a packet with a source IP address identifying the machine program B is running on

■ contain a datagram with a destination port number the same as the source port number in the datagram B received from A

(b) (6 points) Program A sends the following commands in this order:

- to create a file 'foo' with some contents in directory 'bar';
- to list the files in directory 'bar'

Then, it waits replies to these two commands and after waiting for a long time receives only a reply to its first command:

- a list of files in directory 'bar' which does not include 'foo'

Could the file 'foo' have been created? Explain **briefly**.

Solution: Yes; creation reordered after listing, then reply lost/delayed

6. Suppose two programs A and B communicate over a network. They use public-key encryption and digital signatures to have A communicate what packages for B to deliver to third-parties. A sends the following messages, each of which is encrypted with B's public encryption key and signed with A's private signing key. B checks the signatures, decrypts the messages, and acts on them.

   1. deliver a package of type X to C
   2. deliver a package of type Y to D
   3. deliver another package of type X to C
   4. deliver another package of type Y to D

(a) (6 points) An attacker with sufficient control over the network could cause B to send four packages of type X to C despite A not generating encrypted and signed messages to do so. (The attacker has no access to A or B's private keys.) Explain briefly how the attacker would do this.

Solution: repeat third message two times

(b) (6 points) Describe briefly a modification A and B could make to how they communicate that would avoid the scenario in the previous question. Include both any changes to the contents of messages and to how A or B verify the new message contents.

Solution: sequence number on A's messages + B checks that sequence numbers have not been seen before

7. Suppose a system has:

- two-level page tables
- **30**-bit virtual addresses
- 4096-byte (2^{12} byte pages)
- **8-byte** page table entries
- page tables at each level with **512** entries (so page tables take up 4096 bytes)
- a 8-entry, 2-way TLB (translation lookaside buffer)

(a) (8 points) Consider a process accessing the virtual address `0x801090` when the page table base register is `0x400000` (which is a physical byte address, not a physical page number).

Fill in all the blanks below:

To perform this access, first the processor will access a first-level page table entry at physical address

0x400020 .

Then, if that page table entry is valid (and has appropriate permission bits) and contains physical page number `0x9`, the processor will access a second-level page table entry at physical address

0x9008 .

Then, if that page table entry is valid (and has appropriate permission bits) and contains physical page number `0x4` the processor will access data from physical address

0x4090 .

(b) As a result of the access described in the previous question, the TLB will be modified.

i. (3 points) Which set(s) of the TLB will be modified? (Identify the set index(es).)

Solution: 1

ii. (5 points) What (if anything) of the following will be stored in the TLB as a result of the access? **Select all that apply.**

□ the first-level page table entry
□ one or more other entries from the first-level page table
■ the second-level page table entry
□ one or more other entries from a second-level page table
□ the data from the final physical address accessed

(c) (Attributes reproduced from previous page:

- two-level page tables
- **30**-bit virtual addresses
- 4096-byte (2^{12} byte pages)
- **8-byte** page table entries
- page tables at each level with **512** entries (so page tables take up 4096 bytes)
- a 8-entry, 2-way TLB (translation lookaside buffer)

)

Suppose a process on this system has the following memory accessible to it for reading or writing or executing without a page fault or other exception occurring:

- addresses 0x1000–0x4FFF
- addresses 0x6000–0x6FFF
- addresses 0x7FFE000–0x7FFFFFF

i. (4 points) Assuming only the above addresses are accessible, how much memory would need to be allocated to the process's page tables? (Do not include space for the process's data and code.)

Solution: 3 pages = 12Kbytes

ii. (5 points) Suppose the process forks to create a new (child) process and the system's OS uses copy-on-write to implement `fork()`. To save as much space as possible, the OS copies as little as possible as part of its copy-on-write implementation.

Suppose the child process writes 8 bytes to `0x7FFE800` immediately after the fork. Then, ________. **Select all that apply.**

□ the parent (original) and child (new) process may have the same first-level page table

■ the parent and child process will both store the data for virtual address `0x2000` in the same physical address

□ the second-level page table entry for virtual address `0x2000` in the child process will be marked as writable

■ the second-level page table entry for virtual address `0x7FFE000` in the child process will be marked as writable

□ the second-level page table entry for virtual address `0x7FFF000` in the child process will be marked as writable

8. Consider the following C snippets:

```
int global = 0;
void *foo(void *arg) {
    int *p = (int*) arg;
    printf("%d %d\n", global, *p);
    global = *p;
    return NULL;
}

void function1(int v) {
    pid_t pid = fork();
    if (pid == 0) { /* in child process */
        foo(&v);
        exit(0);
    } else {
        waitpid(pid, 0, NULL);
    }
}

void function2(int v) {
    pthread_t p;
    pthread_create(&p, NULL, foo, &v);
    pthread_join(p, NULL);
}
```

Assume none of the library functions called by the code above fail, all relevant header files are included, and that no other relevant code is running during the function calls described below.

(a) (6 points) When `function1` is called, which of the following is true? **Select all that apply.**

- □ when it returns, the value of `global` may have changed
- □ when it returns, a newly created process or thread may be active
- □ it will only return in a new process (with a different pid than when it was called)
- □ the `*p` in the `printf` call in `foo` may read `*p` from another thread's stack
- □ the `*p` in the `printf` call in `foo` could access an out-of-scope value and therefore crash
- ■ if the function is run on a single-core system, then one or more context switches will occur while it runs

(b) (5 points) When `function2` is called, which of the following is true? **Select all that apply.**

- ■ when it returns, the value of `global` may have changed
- □ when it returns, a newly created process or thread may be active
- ■ the `*p` in the `printf` call in `foo` may read `*p` from another thread's stack
- □ the `*p` in the `printf` call in `foo` could access an out-of-scope value and therefore crash
- ■ if the function is run on a single-core system, then one or more context switches will occur while it runs

9. Suppose we have a C program that is built from four files: `utilities.c`, `utilities.h`, `parsing.c`, `parsing.h`, and `main.c`.

Each .c file with a corresponding .h file `#include`s that .h file.

main.c and parsing.c each use functions that are defined in utilities.c and declared in utilities.h, and they `#include`s utilities.h in order to do so.

main.c also uses functions that are defined in parsing.c, and `#include`s parsing.h in order to do so.

(a) (5 points) In a Makefile to build this program, a possible rule might look like:

```
main.o: ???
        clang -Wall -c -o main.o main.c
```

where `???` represents an omitted list of dependencies. Which of the following should be on that list? *-1 point per wrong* **Select all that apply.**

■ `main.c`
□ `main.s`
□ `main.o`
□ `utilities.c`
■ `utilities.h`
□ `utilities.o`
□ `parsing.c`
■ `parsing.h`
□ `parsing.o`

(b) (5 points) When running `make` and the above rule is triggered, `make` will run the command `clang -Wall -c -o main.o main.c`.

Which of the following is true about what happens during this process? **Select all that apply.**

■ the `make` program will make a system call to create a new process
■ the `clang` program will make a system call to open `parsing.h`
□ the command will fail if the current user is not listed as the owner of `main.c`
□ the page table used by `make` will be the same page table used by `clang`
□ the process running `clang` will be the parent of the process running `make`

10. (8 points) Consider the following C code:

```
struct ItemList {
    int key, value;
    struct ItemList *next, *prev;
};

struct HashBucket {
    pthread_mutex_t lock;
    struct ItemList *list_head;
};

struct HashBucket buckets[HASH_SIZE];

int Hash_Move(int from_key, int to_key) {
    struct HashBucket *from = buckets[Hash(from_key)];
    pthread_mutex_lock(&from->lock);
    struct ItemList *item;
    item = FindItemInList(&from->list_head);
    item->key = to_key;
    if (!item) { pthread_mutex_unlock(&from->lock); return 0; }
    struct HashBucket *to = buckets[Hash(to_key)];
    if (to != from) {
        pthread_mutex_lock(&to->lock);
        RemoveItemFromList(&from->list_head, item);
        AddItemToList(&to->list_head, item);
        pthread_mutex_unlock(&to->lock);
    }
    pthread_mutex_unlock(&from->lock);
}
```

Note that this code relies on a hash function called `Hash()` which is not shown, but converts an arbitrary int to a number between 0 and `HASH_SIZE - 1`, inclusive.

If two calls `Hash_Move(A, B)` and `Hash_Move(C, D)` are made simultaneously, it is possible for a deadlock to occur, for certain values of A, B, C, and D, **even if A, B, C, and D are all different from each other**.

Briefly describe what would be true about these distinct values of A, B, C, and D that can trigger the deadlock. (My answer is in the form of a C expression.)

Solution: Hash(A) == Hash(D) and Hash(B) == Hash(C) (and Hash(A) != Hash(B) [no deduction for not including])