CS 3214 Final Exam

This is a closed-book, closed-internet, closed-cell phone and closed-computer exam. However, you may refer to your 2 sheets of prepared notes. **Your exam should have NN pages with 4 topics totaling 150 points. You have 120 minutes**. Please write your answers in the space provided on the exam paper. If you unstaple your exam, please put your initials on all pages. You may use the back of pages if necessary, but please indicate if you do so. Answers will be graded on correctness and clarity. You will lose points if your solution is more complicated than necessary or if you provide extraneous, but incorrect information along with a correct solution.
To be considerate to your fellow students, if you leave early, do so with the least amount of noise.

Name (printed) ____

I accept the letter and the spirit of the Virginia Tech undergraduate honor code – I have not given or received aid on this exam.

(signed) ____

#	Problem	Points	Score
1	Concurrency and Synchronization	40	
2	Memory	40	
3	Communication	50	
4	Essay Question technical content writing	 12 8	
	Total	150	

1. Concurrency and Synchronization (40 points)

a) (5 points) The code below uses the thread safe function `zero_if_found` to operate sequentially on two arrays.

```
int first[N], second[N];

//initialize arrays first and second

void zero_if_found(void* v){
  // change all negative numbers in v to zero
}

zero_if_found(first);           // apply to first array
zero_if_found(second);          // apply to second array
```

Show the pthread code that would be used to operate concurrently on the two arrays.

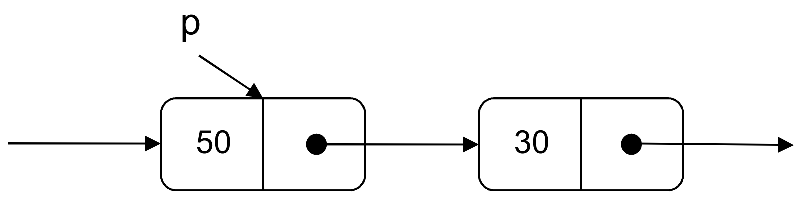
Answer:

```
pthread_t t1, t2;

int pid1 = pthread_create(&t1, NULL, zero_if_found, (void*)first);

int pid2 = pthread_create(&t2, NULL, zero_if_found, (void*)second);
```

b) (10 points)The code shown below for insert_after manipulates a singly linked list by adding a new node with value n after the node pointed to by p. The lines of code are numbered for reference.

```
struct node {
  int value;
  node* next;
}

int insert_after(node* p, int n) {
 if (p == NULL) return 0;                    //line 1
 node* np = (node*)malloc(sizeof(node));     //line 2
 np->value = n;                              //line 3
 np->next = p->next;                         //line 4
 p->next = np;                               //line 5
 return 1;                                   //line 6
}
```

Suppose thread 1 executes the call `insert-after(p, 10)` and thread 2 concurrently executes the call `insert-after(p, 20)`. In each call the value of `p` points to the same node shown in the following diagram.

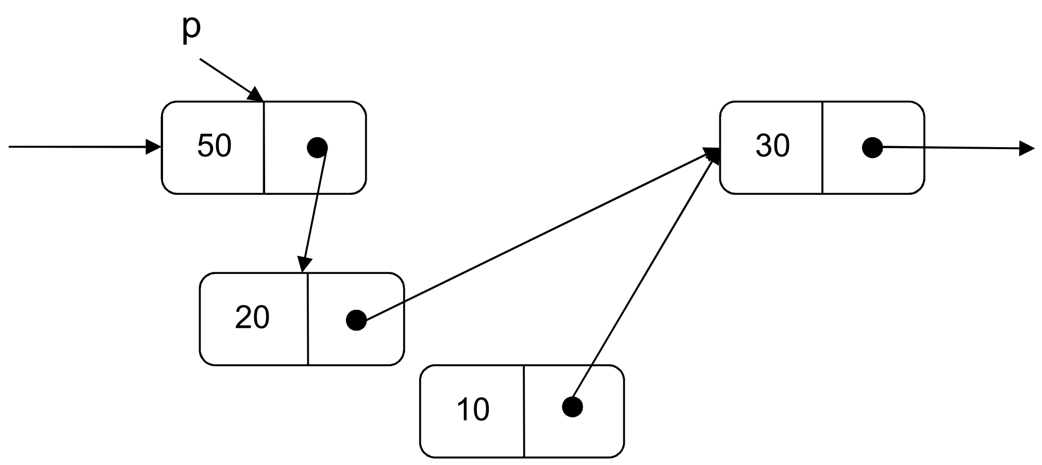


Using the line numbers shown in the code, give a specific concurrent execution sequence of the `insert_after` function by thread 1 and thread 2 that leads to an incorrect result. Also, draw a diagram similar to that above to illustrate the state of the linked list after the incorrect concurrent execution.

Answer: (note – this is one scenario; others are possible)

Thread 1 executes lines 1-4
Thread 2 executes lines 1-4
Thread 1 executes lines 5-6
Thread 2 executes lines 5-6

leading to this incorrect state:



c) (10 points) Is the synchronization for the `insert_after` code shown below correct? If so, explain why it is correct. If not, explain why it is not correct.

```
int insert_after(node* p, int n)
{
 pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

 if (p == NULL) return 0;

 pthread_mutex_lock(mutex);

 node* np = (node*)malloc(sizeof(node));
 np->value = n;
 np->next = p->next;
 p->next = np;

 pthread_mutex_unlock(mutex);
 return 1;
}
```

Answer: this is not correct synchronization because the scope of the lock is local to the insert_after function. This means that the lock is created and destroyed on each invocation of insert_after and thus does not apply across different executions of insert_after by different threads.

d) (15 points) Functions to manage fixed sized blocks of memory are defined as follows. These functions are NOT thread safe.

```
blist* balloc(int nb);  // return a list of nb number of blocks;
                        // return NULL if nb free blocks not
available

void   bfree(blist* bl); // free the blocks in the list bl

int    bavail();        // return current number of free blocks
```

Use these unsafe functions and pthreads synchronization to implement safe block allocator that has the following interface.

```
blist* safe_balloc(int nb);   // is thread safe; is synchronized
                              // with safe_bfree - blocks until
                              // nb blocks are available;

void   safe_bfree(blist* bl); // is thread safe; synchronizes
                              // with save_balloc to indicate
                              // more free blocks available
```

Note: you will need to use both mutual exclusion and condition synchronization. Condition synchronization is needed to insure that `safe_balloc` returns only when the requested number of blocks is available.

Answer:

```
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t avail = PTHREAD_COND_INITIALIZER;

blist* safe_balloc(int nb){
  pthread_mutex_lock(mutex);
  while (nb > bavail())
    pthread_cond_wait(avail);
  blist* result = balloc(nb);
  pthread_mutex_unlock(mutex);
  return result;
}

void   safe_bfree(blist* bl){
  pthread_mutex_lock(mutex);
  bfree(bl);
  pthread_cond_broadcast(avail);
  pthread_mutex_unlock(mutex);
}
```

2. Memory (40 pts)

a) (12 points) Examine the code shown below. Each line of code is numbered by a comment.

```
int x;                            //line 1
int y = 0;                        //line 2
int* xp;                          //line 3

void calculate (int a, int b) {

   xp = malloc(3*sizeof(int));    //line 4
   xp[0] = a + b;                 //line 5
   xp[1] = x + y;                 //line 6

   return;                        //line 7
}
```

During program linking and loading different parts of the above code use or modify different memory sections in the address space. In the following table place an X whenever a line of code is related to the specified memory sections. Note that a given line of code may be related to more than one memory section. Assume parameters are not passed in registers.

Line	BSS	DATA	STACK	HEAP
1	x			
2		x		
3	x			
4	x			x
5	x		x	x
6	x	x		x
7			x	

Answer:

Shown in table above.

b) (8 points) In addition to those sections listed in the table in part (a), the address space has two additional regions that have well defined purposes. Identify and briefly describe these three regions.

Answer:

1. The text region that contains the executable code.
2. The shared library/mmap region which contains shared library code and memory explicitly allocated through the mmap system call.
3. The kernel region which contains code for the kernel that is part of the process address space.

c) (5 points) A process has just done an exec system call to load a program that is statically linked to a commonly used library (e.g., libc.a). Is it highly likely or highly unlikely that the first reference at run-time to a member of this library will cause a page fault? Explain.

Answer: Highly likely because the page containing the library code has probably not been referenced previously. Thus, a page fault will be encountered on the first reference to load the page into memory.

d) (5 points) A process has just done an exec system call to load a program that is dynamically linked to a commonly used library (e.g., libc.so). Is it highly likely or highly unlikely that the first reference at run-time to a member of this library will cause a page fault? Explain.

Answer: Highly likely because the code, though resident in main memory, has not yet been mapped into the address space of the calling process.

e) (10 points) A computer has a single level cache, main memory, and a disk used for virtual memory. If a referenced word is in the cache, 20 ns are required to access it. If it is in main memory but not in the cache, 60 ns are needed to load it into the cache, and then the reference is started again. If the word is not in main memory, 12 milliseconds are required to fetch the word form disk, followed by 60 ns to copy it to the cache, and then the reference is started again. The cache hit ratio is 0.9 and the main-memory hit ratio is 0.6.

Show a complete <u>formula</u> for the average time in ns required to access a referenced word on this system? Do not reduce/evaluate the formula.

Answer:
cache access time is 20 ns
memory access time is 80 ns
disk access time is $12x10^6 + 80$ ns

average access time: $0.9*20ns + 0.1*(0.6*80ns + 0.4* (12x10^6 + 80ns))$

3. Communication (50 points)

a) (14 points) With respect to project 5, your solution could have employed a thread pool or spawned threads on demand. Which option would you consider more appropriate for that particular application? Discuss your rationale for the approach that you chose.

Answer: Here are some points both ways.

Points favoring using a threadpool:

- Creating a new thread can be (relatively) expensive; using a thread pool allows us to reuse a single thread for a succession of tasks (connections), and only pay the creation cost once per thread.
- Paying the cost of thread creation all at once during startup will provide for better performance afterwards.
- We expect typical tasks for the threads (connections) to be short-lived, hence a poor ratio between thread creation time and thread execution time.

Points favoring threads-on-demand:

- We expect typical tasks for the threads (connections) to be long-lived, hence a good ratio between thread creation time and thread execution time.
- We expect the number of concurrent connections to be small (and long-lived), and do not want to incur overhead of unutilized threads in the pool.
- Using a static-sized thread pool would limit the number of threads available to handle tasks (connections), and could lead to poor response times, if the number of connections was large.

b) (20 points) Consider the following flawed attempt at implementing the sysstatd web service in Project 5.

```
int main(int argc, char *argv[]) {

    struct addrinfo *ai = NULL;
    struct addrinfo hints;
    int socket = -1;

    hints.ai_flags = AI_PASSIVE | AI_ADDRCONFIG;
    hints.ai_socktype = SOCK_STREAM;

    int e = getaddrinfo(NULL, "html", &hints, &ai);
    if (e != 0)
      error(EXIT_FAILURE, 0, "getaddrinfo: %s", gai_strerror(e));

    // For simplicity, we will assume we are only responsible
    // for a single address here.

    int listenfd = -1;
    listenfd = socket(ai->ai_family, ai->ai_socktype,
                      ai->ai_protocol);
    if (listenfd == -1)
       error (EXIT_FAILURE, errno, "listenfd");
    int opt = 1;
    setsockopt(socket, SOL_SOCKET, SO_REUSEADDR, &opt,
               sizeof (opt));

    if (listen(listenfd, SOMAXCONN) != 0)
       error(EXIT_FAILURE, errno, "listen");

    struct sockaddr clientaddr;
    socklen_t clientaddrlen = sizeof(clientaddr);

    while ( (socket = accept(listenfd, &clientaddr,
                                     &clientaddrlen)) != -1) {
       char buf[1024], *url;
       int rc = read(socket, buf, 1024);
       parse_http_request(buf, &url);
       if (strcmp(url, "/loadavg") == 0) {
          char reply[1024];
          char *jsonMsg = read_json("/loadavg");
          snprintf(reply, sizeof reply,
              "HTTP/1.1 200 OK\r\nContent-Length: %d\r\n\r\n",
              strlen(jsonMsg));
          write(socket, reply, strlen(reply));
```

```
        } else {
           send_404(socket);
        }
     }
     close(socket);
    }
```

The code contains multiple errors related to the use of TCP sockets and to the implementation of the HTTP protocol. Identify five such mistakes. (Assume that the chosen buffer sizes are adequate, and that code not shown is implemented correctly.)

Answer: There were a number of errors, including:

- left members of the struct hints uninitialized
- called setsockopt() on socket instead of listenfd
- did not call bind()
- did not deallocate *ai
- did not write jsonmsg to the socket
- close(socket) should be inside the while loop

c) (16 points) When using sockets, what purpose is served by calls to each of the following, and would the call be made by the client or server or both:

(1) bind()

Answer: Calling bind() assigns an address (specified by a parameter to bind()) to a socket.
Called by server.

(2) listen()

Answer: Calling listen() marks the socket as passive, i.e., one that will be used to accept() incoming connection requests.
Called by server.

(3) connect()

Answer: Calling connect() connects a socket to a specific address.
Called by client.

(4) accept()

Answer: Calling accept() creates a new connected socket (and returns a new file descriptor referring to that socket) for the first pending connection on a "listening" socket.
Called by server.

4. Essay Question (20 points)

Identify the tradeoffs involved between programmer-managed dynamically allocated memory (as in malloc/free) versus automatically managed dynamic memory (as in garbage collection). Be specific about the tradeoffs that you identify and use appropriate technical vocabulary.

Note: *This question will be graded both for technical content of your arguments (12 points) and for your ability to communicate effectively in writing (8 points). Your answer should be well-written, organized, and clear.* ***Your answer must be legible – points will be deducted for parts that cannot be read with normal effort. Use the back of this page as needed.***

Grading:

Technical Content:

- describes programmer-managed memory as giving greater application control or efficiency but allowing greater possibility of errors
- describes possible errors as memory leaks, releasing memory too soon
- describes value of application control or gives measures of efficiency (e.g., better memory utilization)
- describes garbage collection as preventing classes of memory allocation errors but requires higher overhead
- describes higher overhead as additional memory and additional time to perform garbage collection
- describes appropriate use of technical terminology

Writing:

- has well structured sentences
- uses appropriate word choice and language
- has good ordering/clarity of ideas presented
- has good coherence and understandability
- overall impression

Technical Content: __________ Writing: __________