CS360 Final 2021

Instructions: Please answer all questions. If it matters and it is not specified, your machine has four-byte pointers and is little endian. In case you need them, here are prototypes:

```
int pthread_create(pthread_t *thread, const pthread_attr_t *attr, void *(*start_routine)(void *), void *arg);

int pthread_mutex_init(pthread_mutex_t *mutex, const pthread_mutexattr_t *attr);
int pthread_mutex_lock(pthread_mutex_t *mutex);
int pthread_mutex_unlock(pthread_mutex_t *mutex);

int pthread_cond_init(pthread_cond_t *cond, const pthread_condattr_t *attr);
int pthread_cond_wait(pthread_cond_t *cond, pthread_mutex_t *mutex);
int pthread_cond_signal(pthread_cond_t *cond);

int scanf(const char *restrict format, ...);   /* Returns the number of successful matches it made. */

int fork();
int execlp(const char *file, const char *arg0, ... /*, (char *)0 */);
int wait(int *stat_loc);
int dup2(int fildes, int fildes2);
int pipe(int fildes[2]);
int close(int fildes);
int kill(pid_t pid, int sig);

unsigned alarm(unsigned seconds);

typedef void (*sig_t) (int);
sig_t signal(int sig, sig_t func);
```

Question 1 - 25 points

This one came from a bank -- here is a representative question:

For this question, please assume that malloc() is implemented as described in lecture, with four-byte pointers, and free list structures composed of size, flink and blink, in that order. You are given the following facts:

- The heap starts at address 0x147d58
- The heap has 512 bytes.
- There are exactly four allocated memory chunks.
- The free list is composed of the following nodes in this order:

```
Address: 0x147e18   Size: 32 (0x20)
Address: 0x147e58   Size: 48 (0x30)
Address: 0x147db8   Size: 32 (0x20)
Address: 0x147d58   Size: 40 (0x28)
```

Please answer the following questions. Please put your answer to each question on its own line, in the format `question number: answer`. Also, if your answer is in hex, please precede it with '0x'

1: Consider the allocated memory chunk with the smallest address (not the smallest size, but the smallest address). What is the starting address of that chunk?

2: Consider the allocated memory chunk with the largest address (not the largest size, but the largest address). What is the starting address of that chunk?

3: What is the value at address 0x147d58?

4: What is the value at address 0x147e5c?

5: What is the value at address 0x147e60?

6: There is an allocated memory chunk that begins at address 0x147dd8, and is 64 bytes in size. When the user called malloc() and got this chunk, what value was returned to the user?

7: When the user called malloc() and got this chunk, tell me a legal value of the argument that the user could have given to the malloc() call.

8: If I call sbrk(0), it will return 0x147f58. If I try to write to this address, I won't get a seg fault? Why? Make your answer fit on one line.

Question 2 - 5 points

Write a C program where you create an orphan process that prints the line: "I am an orphan." When it prints that line, it should be an orphan process. If you want a process to delay, you should have it call **sleep()**.

Question 3 - 5 points

In two to four sentences, define a Zombie process. Be precise.

Question 4 - 10 points

Write a program in C which creates a new process. Have that new process generate SIGPIPE, which will kill it, and then have the original process return. Don't bother with include files. As always, if you're unsure that you're communicating correctly in your program, describe what your program does in English, and you'll get a few points of partial credit. To help you, make sure you set your "paragraph" setting to "preformatted". You may only use the following system or library calls.

```
int close(int fildes);
int fork();
int pipe(int fildes[2]);
int write(int fildes, const void *buf, size_t nbyte);
int wait(int *status);
```

Question 5 - 5 points

This one came from a bank -- here's an example:

How many lines are printed by the following program:

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

int main()
{
  char *s = "Luther";
  char *x;

  for (x = s; *x != '\0'; x++) fork();
  printf("Fred\n");
  return 0;
}
```

Question 6 - 25 points

You've just starting a new job at a company that is developing its own cryptography algorithms. Your co-worker, whose name happens to be "Sleven", has written two programs called "**slencode**" and "**sldecode**". **Slencode** encodes. **Sldecode** decodes. Your job is to stress test them. You don't have the source code -- just the executables. And Sleven has gone on vacation. What his programs do is as follows:

- Neither of them uses any command line arguments.
- Each of them reads input from file descriptor 7. Why 7? Because he's Sleven.
- Each prints output on standard output.

You're going to stress test them by writing a C program that makes them work together. Specifically -- you want **slencode** to read from standard input, and you want **sldecode** to read from the output of **slencode**. Your program should either return when both **slencode** and **sldecode** are finished, or if ten seconds have passed, in which case both **slencode** and **sldecode** should be terminated, and your program should return.

Now, you're not really writing this program, but instead you're going to be answering questions about it. When we run it, there will be three processes, which we will name "E" for **slencode**, "D" for **sldecode**, and "Y" for yours. You should assume that there is a variable "int p[2]" which is used for a pipe. If you want to use a second pipe, then use a variable "int p2[2]".

As always, in your answer box, please put each answer on one line, preceded with its question number. If there are multiple answers to a question, or the question calls for a one to two sentence answer, please put that all on one line, preceded with the question number.

Question 1: Which process(es) calls **pipe**()?

Question 2: How does the "E" process know that it's the "E" process?

Question 3: Exactly what **dup2**() calls are made by the Y process?

Question 4: Exactly what **dup2**() calls are made by the E process?

Question 5: Exactly what **dup2**() calls are made by the D process?

Question 6: Exactly what **close**() calls are made by the E process?

Question 7: Exactly what **execlp**() call is made by the E process?

Question 8: How would you use threads in this problem? Please give me a two sentence answer.

Question 9: If you didn't use threads, explain how you would use a signal handler and/or the alarm system call for this problem? Please give me a one to three sentence answer.

Question 7 15 points

Below is a threaded program with two parts missing. In the answer box below, write the missing parts. Label the first part as "/* **Missing Thread Code** */" and the second part as "/* **Missing Main Code** */" Here's what the program is going to do:

- The main() thread creates some shared and per-thread data, and creates n threads.
- The main() thread goes into a loop where it:
    - Prints the current time since the beginning of the program, along with "main"
    - Sleeps for a second
    - Wakes up thread i using a condition variable specific to thread i
    - Waits for thread i to wake it up using the "main_cv" condition variable
- It starts with i = 0, and increments each iteration to n-1, and then starts over at i = 0.
- Each thread goes into a loop where it does the following:
    - It waits on its condition variable.
    - When it wakes up, it prints the current time and its thread id.
    - It then reads an integer from standard input, and sleeps for that many seconds.
    - It then wakes up the main thread using the "main_cv" condition variable.
- When a thread reads EOF, or it fails to read an integer, the program should end.

Here's an example:

```
UNIX> cat input.txt
2
1
5
1
4
UNIX> ./a.out 3 < input.txt
  0: Main
  1: Thread 0
  3: Main
  4: Thread 1
  5: Main
  6: Thread 2
 11: Main
 12: Thread 0
 13: Main
 14: Thread 1
 18: Main
UNIX>
```

And here's the code. BTW, this is not a useful program. I'm just wanting you to demonstrate to me your understanding of the synchronization primitives in pthreads.

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

struct Shared {
  pthread_mutex_t *lock;
  pthread_cond_t **cvs;
  pthread_cond_t *main_cv;
  long base_time;
};

struct Private {
  int id;
  struct Shared *s;
};

void *thread(void *a)
{
  struct Private *p;
  struct Shared *s;
  int num;

  p = (struct Private *) a;
  s = (struct Shared *) p->s;

  /* Missing Thread Code */
}

int main(int argc, char **argv)
{
  pthread_t *tids;
  struct Shared *s;
  struct Private *p;
  int i;
  int n;
  void *v;

  if (argc != 2)  { fprintf(stderr, "usage: tp num-threads.\n"); return 1; }

  n = atoi(argv[1]);
  tids = (pthread_t *) malloc(sizeof(pthread_t) * n);
  s = (struct Shared *) malloc(sizeof(struct Shared));
  s->base_time = time(0);

  s->lock = (pthread_mutex_t *) malloc(sizeof(pthread_mutex_t));
  pthread_mutex_init(s->lock, NULL);
  s->main_cv = (pthread_cond_t *) malloc(sizeof(pthread_cond_t));
  pthread_cond_init(s->main_cv, NULL);

  s->cvs = (pthread_cond_t **) malloc(sizeof(pthread_cond_t *) * n);

  for (i = 0; i < n; i++) {
    s->cvs[i] = (pthread_cond_t *) malloc(sizeof(pthread_cond_t));
    pthread_cond_init(s->cvs[i], NULL);
  }

  for (i = 0; i < n; i++) {
    p = (struct Private *) malloc(sizeof(struct Private));
    p->id = i;
    p->s = s;
    pthread_create(tids+i, NULL, thread, (void *) p);
  }

  /* Missing Main Code */

}
```

Question 8 - 10 points

Now, I want to you start with the program above again (as if you never answered the previous problem) and assume that we have added an integer variable named n to the Shared data structure. Your job is to once again write "/* **Missing Thread Code** */" and "/* **Missing Main Code** */". Here's what the program should do:

- The main() thread should block until all of the other threads are "ready".
- All of the other threads should print out "I'm ready", to signify that they are ready. When a thread is ready, it should simply go into a "while(1)" loop that does nothing.
- When all of the other threads are "ready", the main() thread should print "I'm Ready Too", and then go into a while(1) loop.

Again, this is not a useful program -- I'm just testing you with thread synchronization.

Example:

```
UNIX> ./a.out 4
I'm Ready
I'm Ready
I'm Ready
I'm Ready
I'm Ready Too                          # At this point, all five threads are spinning on while(1) loops.
```