SY204 – Sys Prgm & OS Fund

Name(s): McKenzie Eshleman

Alpha(s): 221938

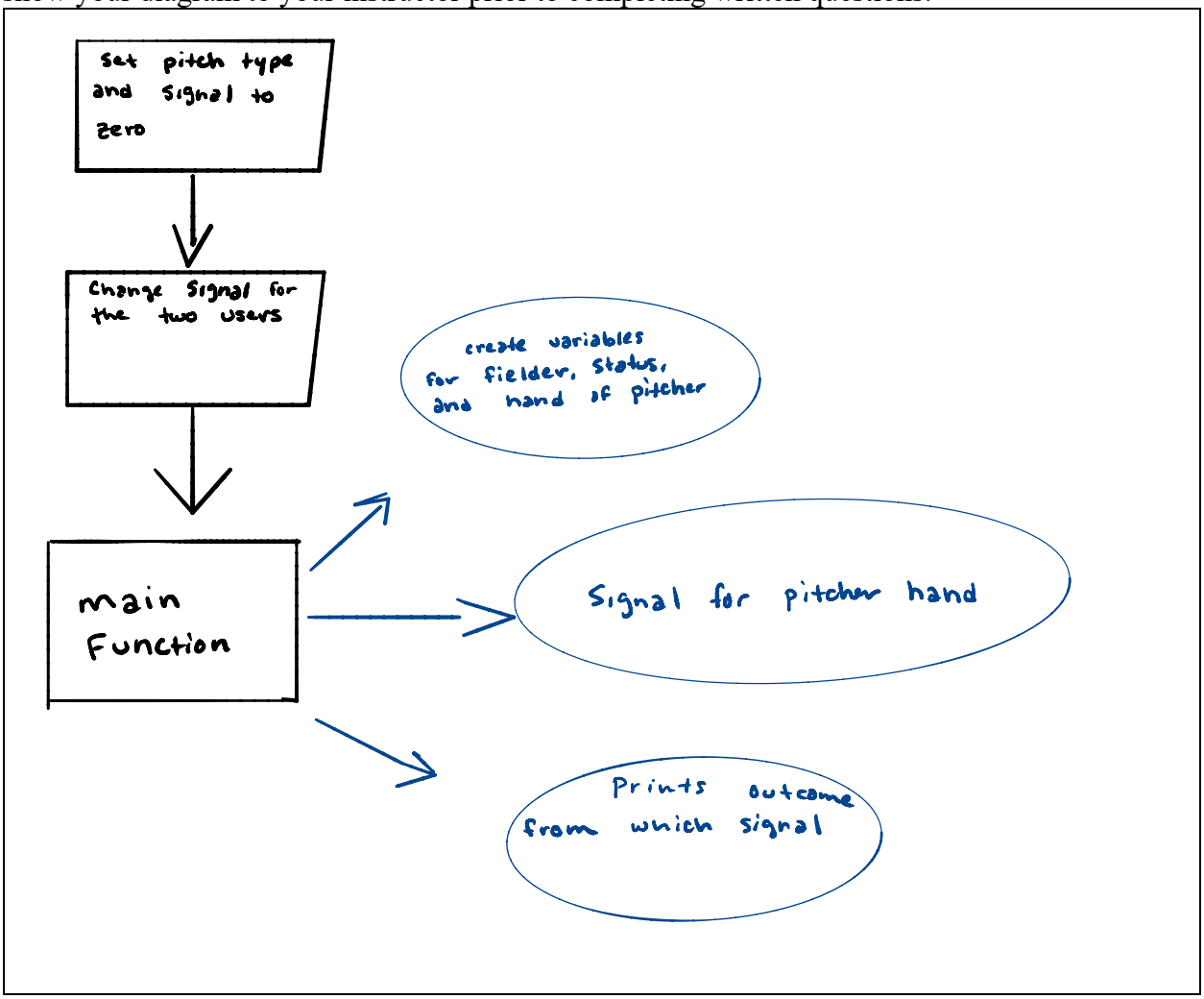
Assignment Type:	Programming – Laboratory	Collaboration Policy:	Default
Assignment Title:	Baseball		

1. General (5).

See assignment webpage for due dates.

a. Description. Read through this entire assignment before starting work. In this assignment you will create a program that catches signals and performs actions based on the caught signal (output a message, send a signal). Eventually, your program will additionally send a signal back to the process that sent the initial signal. Your program will accept signals until stopped via a signal.

b. [5 / 0] After reading through the assignment at least twice, draw a high level (initial, rough idea) block diagram that depicts the ordering of the tasks of Implement Part 1 – Play Ball. Your diagram should be neatly hand drawn, but is not expected to be immaculate. You *shall* show your diagram to your instructor prior to completing written questions.



Instructor Initials: _____

2. Research (10).

a. [3 / __ / 0] Read *LPI* Chapter 21 Section 21.1.3, and complete the following.

What C reserved word should be used for variables modified by a signal handler and another function within the program? volatile

What data type should be used as a flag between a signal handler and another function within the program? Signal handlers

b. [3 / __ / 0] Continuing from above, complete the following. Scenario: You are developing code that you understand will be used for years to come.

What are the minimum and maximum values you should assign to a variable that will be shared between a signal handler and other functions?

-127	127
0	255
Min	Max

Explain why you chose those values. In other words, what design principle are you adhering to? Hint: What is the minimum size of the data type from 2.a. per POSIX?

The standards require that the range be -127 to 127 if the signal is represented by a signed value, or 0 to 255 if it is represented as a unsigned value.

c. (4) Read *LPI* Chapter 22 Section 22.2, and complete the following.

[2 / __ / 0] When will `SIGCONT` always cause a process to continue execution?

If a process is currently stopped, SIGCONT always causes the process to resume, even if the process is currently blocking or ignoring.

[2 / __ / 0] Paragraph 4.g. Part 3 – Batter's Up has you implement the sending side of a simple heartbeat signal. In your own words describe why a process listening for a heartbeat signal from another process should not rely on missing a single, or a low number of, heartbeat signal(s) before taking recovery actions.

because the heart beat signal is not always constant.

3. Plan (5).

a. [1 / 0] Read *LPI* Chapter 21 Section 21.1.2 and answer the following based on the requirements discussed in Paragraph 4.d. Safety First. In your own words describe what should be done to ensure a signal handler is safe/reentrant if external (to the signal handler) routines are to be called from within the signal handler. Discuss a specific example from the below example signal handler. Hint: The below signal handler is not safe, what should be done to make it safe.

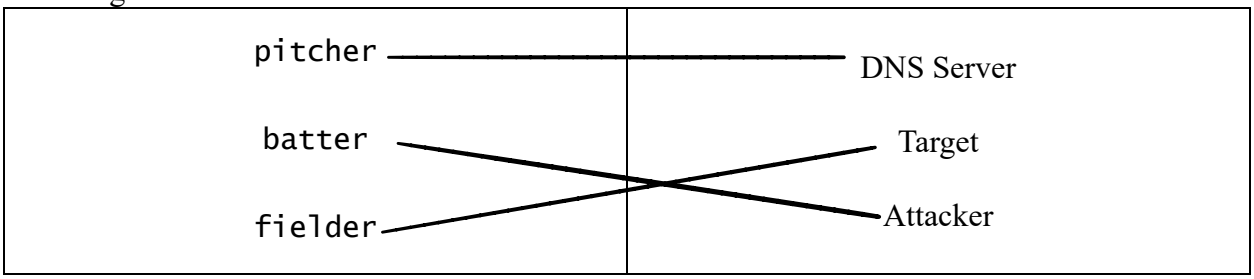
```
void simpSigHand(int intSigNum) {
  write(STDOUT_FILENO, "Nothing can stop me now\n", 24);  // SAFE line
  return ;
}
```

Change STDOUT_FILENO TO STDIN

b. [1 / 0] Answer the following based on the requirements discussed in Paragraph 4.d. Safety First. Describe how you are plan to implement the output requirements from Paragraph 4.e. Part 1 – Play Ball, specifically discuss what kind of I/O you are going to use (Descriptor I/O or C Library I/O) and where you plan to have the I/O operations performed.

I will use a C Library I/O To preform the operations

c. [1 / __ / 0] Read the US-CERT Alert TA13-088A, referenced on the assignment web page. Match the lab program on the left to the respective label from a DNS Amplification Attack on the right.



d. [2 / __ / 0] Continuing from US-CERT Alert TA13-088A, in your own words describe how a working version of `batter` could be used as a form of redirection to attack a target on a local system.

The batter could send a signal that redirects to a attack.

4. Implement (40).

a. [0 / -5 / -10] Sound and Secure Cyber Design

- You shall minimize the use of global variables other than `errno`, `opt*`, and for error reporting.
- You may use global variables related to signal handling routines, but must do so *safely*.
  - Hint: Review your research.
- You are encouraged to use `#define`, and other preprocessing directives.
  - Your program will not be compiled using the `-d` compiler directive.
- Source code is insufficiently documented: complex code blocks not explained, lack of robust program and function descriptions (readability).
- Source code is poorly or inconsistently formatted, but can be compiled (readability).
- Source code uses poorly named functions or poorly named variables (readability).
- Source code does not use associated common libraries (reusability).
- Source code does not use library defined constants (portability).
- Source code does not use deprecated language features (design for the future).
- Implementation does not violate Principle of Least Privilege.

b. [0 / -5] Usage Option (All Parts)

- Read in the "`-h`" (help) option using `getopt(3)`, that does not take any arguments.
- Upon detection of the "`-h`" option display simple usage information that describes how to use your program. Example: program name, listing and discussion of command line arguments and options.
- You may assume that all options will appear before arguments.
- If no command line arguments are passed in, output the usage message.
- You may use C Library I/O routines for this feature.
- After you output the usage message, exit the program with a normal return value.
- Implement the outputting of the usage message as a function your program calls.

c. [5 / __ / 0] Error Handling (All Parts)

- For an error that occurs from a system call, output a simple error message (recall `perror(3)`), and exit the program returning an error number.
- At least return a different error number for each of the systems calls that you use. That is, multiple calls to the same system call need not return unique error numbers, but calls to different system calls shall return unique error numbers.
- You may use C Library I/O routines for reporting errors.
- You *do not* need to handle errors from outputs to standard output or standard error, but you *should not* assume that errors will not occur from writes to standard I/O streams.
- If an incorrect option or too many arguments are entered, then output the usage statement and exit the program returning a unique error number.
- Output error messages to standard error.

d. [5 / __ / 0] Safety First (All Parts)

- Design your signal handler(s) such that they are safe (per SUSv3) and reentrant. Hint: Review *LPI* Chapter 21 Section 21.1.2.

e. [10 / __ / 0] Part 1 – Play Ball

- Create signal handling functionality that catches standard user defined signals.
  - For `SIGUSR1`: Output to Standard Output "Fly Ball to: *PID*", where PID is the PID of the process specified via the first non-option command line argument. Additionally, send `SIGUSR1` to the process specified via the first non-option command line argument.
    Note: Part 1 does not require any command line options.
  - For `SIGUSR2`: Output to Standard Output "Ground Ball to: *PID*", where PID is the PID of the process specified via the first non-option command line argument. Additionally, send `SIGUSR2` to the process specified via the first non-option command line argument.
  - You may use Descriptor I/O or C Library I/O routines, but you must use them *safely*.
- Your program shall stop upon receiving any signal whose default disposition terminates the program. Note: This does not mean you have to catch any additional signals.
- Your program *may* output its own PID to standard error using C Library I/O routines; the provided `batter` programs do this.

f. [10 / __ / 0] Part 2 – Switch Hitter

- Complete all of Part 1 before beginning Part 2.
- Accept as a command line option "`-r`" (right-handed) specifying that your program will perform as specified in Part 1.
- Accept as a command line option "`-l`" (left-handed) specifying that your program will perform in the following ways:
  - For `SIGUSR1`: Output to Standard Output "Ground Ball to: *PID*", where PID is the PID of the process specified via the first non-option command line argument. Additionally, send `SIGUSR1` to the process specified via the first non-option command line argument.
  - For `SIGUSR2`: Output to Standard Output "Fly Ball to: *PID*", where PID is the PID of the process specified via the first non-option command line argument. Additionally, send `SIGUSR2` to the process specified via the first non-option command line argument.
- You may assume that only a single option "`-r`" or "`-l`" will be given, and that any option specified will appear before the PID argument.
- If no command line option is specified, your program shall perform as specified in Part 1.

g. [10 / __ / 0] Part 3 – Batter's Up

- Complete all of Part 2 before beginning Part 3.
- After a signal is received, send `SIGUSR1` to the process that your process received the signal from.
- Note: Congratulations! You've made it to the majors, test your batter against Nolan Ryan (`pitcherNolan`). Nolan will wait for your signal.

5. Test (40).

a. [0 / -20] Code Compilation and Running

- Source code compiles (`gcc -Wall ...`) under Ubuntu 18.04 LTS x86-64 with no warnings and no errors.
- Compiled program runs without crashing or hanging on test cases (no SEGFAULTs, no infinite loops).

b. [3 / __ / 0] Usage Option (All Parts)

- Solution correctly outputs a usage message when the help option is entered.
- Solution correctly outputs a usage message when no command line arguments are entered.
- Solution correctly returns a normal exit value.

c. [2 / __ / 0] Error Handling (All Parts)

- Solution correctly outputs error messages to standard error.
- Solution correctly returns unique values for different system calls and per specification.

d. [15 / __ / 0] Part 1 – Play Ball

- Solution correctly responds to `SIGUSR1`.
- Solution correctly responds to `SIGUSR2`.

d. [15 / __ / 0] Part 2 – Switch Hitter

- Solution correctly responds when right-handed option is specified.
- Solution correctly responds when left-handed option is specified.

e. [5 / __ / 0] Part 3 – Batter's Up

- Solution sends `SIGUSR1` to the correct process.

6. Submit. Turn in this worksheet and submit source code per the following specifications.

- Turn in this worksheet to your instructor after you have submitted your final source code.
- *COVID-19*: Submit your initial block diagram as a simple screen shot as a single file named `batter-block`, using the standard extension for the file type you submit (e.g. `.pdf`, `.png`) following the submission directions on the course information web page.
- Submit your final code as a single file named `batter-#.c`, where # is the part of the implementation that you completed (e.g. `batter-3.c`), following the submission directions on the course information web page.
- `submit` Assignment (project): *lab05*

[0 / -10 / -15 / -20] Submitted source code that demonstrated insufficient effort (lack of attempt to solve problem)
[0 / -5] Submitted assignment using incorrect assignment type or incorrect assignment number.
[0 / -5] Submitted incorrect number of files in submission directory.
[0 / -5] Submitted file(s) named incorrectly.
[0 / -5] Submitted file(s) do not include name and alpha.