**CS 551 Systems Programming, Summer 2024**
Homework Assignment 3

Out: 7/26/2024 Fri.
**Due: 7/31/2024 Wed. 23:59:59**

---

## Q&A (100 points)

**I.1.** (40 points) The following code is an implementation of the `abort()` function as specified by POSIX.1 (the code is shown as the Figure 10.25 in "Advanced Programming in the UNIX Environment").

```
1 void abort(void)
2 {
3     sigset_t
4     struct sigaction    action;
5
6     /* Caller can't ignore SIGABRT, if so reset to default */
7     sigaction(SIGABRT, NULL, &action);
8     if (action.sa_handler == SIG_IGN) {
9         action.sa_handler = SIG_DFL;
10         sigaction(SIGABRT, &action, NULL);
11     }
12     if (action.sa_handler == SIG_DFL)
13         fflush(NULL);           /* flush all open stdio streams */
14
15     /* Caller can't block SIGABRT; make sure it's unblocked */
16     sigfillset(&mask);
17     sigdelset(&mask, SIGABRT);  /* mask has only SIGABRT turned off */
18     sigprocmask(SIG_SETMASK, &mask, NULL);
19     kill(getpid(), SIGABRT);    /* send the signal */
20
21     fflush(NULL);               /* flush all open stdio streams */
22     action.sa_handler = SIG_DFL;
23     sigaction(SIGABRT, &action, NULL);      /* reset to default */
24     sigprocmask(SIG_SETMASK, &mask, NULL);  /* just in case ... */
25     kill(getpid(), SIGABRT);                /* and one more time */
26
27     exit(1);    /* this should never be executed ... */
28 }
```

(1) Describe what a `abort()` library function is supposed to do, and the default disposition of the signal `SIGABRT`

(2) Why does the implementation call "`kill(getpid(), SIGABRT)`" twice (i.e., line 19 and line 25)? Can we replace line 21 to line 27 with a simple `exit()`?

(3) What is the reason that the line 27 "should never be executed"?

(4) The implementation blocks all the signals but SIGABRT (using `sigprocmask()`) before each `kill()`. Do we need to unblock other signals (either inside or outside of the above code)? Why?

**I.2.** (20 points) Read the following code and answer the questions that follow.

```
1 pid_t cpid_1, cpid_2;
2 cpid_1 = fork();
3 if (cpid_1 == -1) {
4     printf("fork 1 failed.\n");
5     exit(1);
6 }
7 else if (cpid_1 == 0) { /* child */
8     cpid_2 = fork();
9     if (cpid_2 == -1) {
10         printf("fork 2 failed.\n");
11         exit(1);
12     }
13     else if (cpid_2 == 0){ /* grandchild */
14         ... ...   /* do real work here */
15         exit(0);  /* after doing real work */
16     }
17     else{
18         exit(0);  /* make grandchild an orphan */
19     }
20 }
21
22 while (1) {
23      pid_t res = waitpid(cpid_1, &status, WNOHANG);
24      if (res == -1) {
25          perror("waitpid");
26          exit(EXIT_FAILURE);
27      }
28      .../* parent carries on to do other things */
29 }
```

(1) Describe what the above code does, and the outcome of the code.

(2) In what circumstances might the code be useful?

**I.3.** (40 points) The following code shows a typical application scenario of conditional variables.

```
1 struct msg {
2     struct msg *m_next;
3     /* ... more stuff here ... */
4 };
5
6 struct msg *workq;
7 pthread_cond_t qready = PTHREAD_COND_INITIALIZER;
8 pthread_mutex_t qlock = PTHREAD_MUTEX_INITIALIZER;
9
10 void process_msg(void)
11 {
12     struct msg *mp;
13
14     for (;;) {
15         pthread_mutex_lock(&qlock);
16         while (workq == NULL)
17             pthread_cond_wait(&qready, &qlock);
18         mp = workq;
```

```
19          workq = mp->m_next;
20          pthread_mutex_unlock(&qlock);
21          ... ... /* process the message mp */
22      }
23  }
24
25  void enqueue_msg(struct msg *mp)
26  {
27      pthread_mutex_lock(&qlock);
38      mp->m_next = workq;
39      workq = mp;
30      pthread_mutex_unlock(&qlock);
31      pthread_cond_signal(&qready);
32  }
```

1. If mutex is the only thread synchronization method allowed to be used, rewrite the `process_msg()` function and the `enqueue_msg()` function (we can assume a pthread function always returns successfully, so no need to check the return value).
2. Describe why using conditional variable is a favorable solution in this scenario.
3. Describe what happen to the consumer thread (i.e. the thread running `process_msg()`) and the mutex `qlock`, when `pthread_cond_wait()`/`pthread_cond_signal()` are called.
4. In the `enqueue_msg()` function, can we switch the order of line 30 and line 31 (i.e., signaling the conditional variable before unlocking the mutex `qlock`)?

---

**Submission instructions**

- Type your answers using whatever text editor you like, remember to include the index number of each question.
- Export the file to PDF format.
- Name the PDF file based on your BU email ID. For example, if your BU email is "abc@binghamton.edu", then the PDF file should be named as "hw3_abc.pdf".
- Submit the PDF file to Brightspace website before the deadline.

---

**Grading guidelines**:

(1) If the submitted PDF file/ZIP file/included in the ZIP file are not named as specified above (so that it causes problems for TA's automated grading scripts), 10 points off.

(2) If the submitted code does not compile:

```
1   TA will try to fix the problem (for no more than 3 minutes);
2   if (problem solved)
3     1%-10% points off (based on how complex the fix is, TA's discretion);
4   else
5     TA may contact the student by email or schedule a demo to fix the problem;
6     if (problem solved)
7       11%-20% points off (based on how complex the fix is, TA's discretion);
8     else
9       All points off;
```

(3) Lastly but not the least, stick to the collaboration policy stated in the syllabus: you may discuss with your fellow students, but code should absolutely be kept private.