Systems Programming (2024 Fall)
Handwritten Assignment 3

B12902110 呂承諺

1. We still use `fork` if we want to spawn a child process that eventually calls `exec` and runs a different executable. Otherwise, the parent process itself would be replaced.

   Moreover, if we want the new process to have its own address space (e.g., heap, data, bss) and file descriptors, we would use `fork()`.

2. If a `SIGALRM` is sent to the program from outside before `setjmp(env_alrm)` is called, then `env_alrm` would be undefined.

   Also, it's better to use `sigsetjmp()` and `siglongjmp()`, otherwise `SIGALRM` may remain blocked after `longjmp()` from `sig_alrm()` back to `main()`.

3. (A) `pthread_cond_wait(&q->not_full, &q->mutex)`

   (B) `pthread_cond_singal(&q->not_empty)`

   (C) `pthread_cond_wait(&q->not_empty, &q->mutex)`

   (D) `pthread_cond_singal(&q->not_full)`

   We need the checks otherwise a deadlock would happen. Both functions would wait for the other's signal before signaling.

4. Just like the example in the textbook.

```c
#define NLOOPS 1000
/* size of shared memory area */
#define SIZE sizeof(long)

static int update(long *ptr) {
  /* return value before increment */
  return ((*ptr)++);
}

int main(void) {
  int i, counter;
  pid_t pid;
  void *area;
  if ((area = mmap(0, SIZE, PROT_READ | PROT_WRITE, MAP_ANONYMOUS | MAP_SHARED,
                   -1, 0)) == MAP_FAILED)
    err_sys("mmap error");

  TELL_WAIT();
  if ((pid = fork()) < 0) {
    err_sys("fork error");
  } else if (pid > 0) { /* parent */
    for (i = 0; i < NLOOPS; i += 2) {
      if ((counter = update((long *)area)) != i)
        err_quit("parent: expected %d, got %d", i, counter);
      TELL_CHILD(pid);
      WAIT_CHILD();
```

```
    }
  } else { /* child */
    for (i = 1; i < NLOOPS + 1; i += 2) {
      WAIT_PARENT();
      if ((counter = update((long *)area)) != i)
        err_quit("child: expected %d, got %d", i, counter);
      TELL_PARENT(getppid());
    }
  }
  return 0;
}
```