Homework 4: Locking, fork and pipes

Answer the following questions.

1 Locks

1.

```
struct list *ready_list; // global
struct thread *cur_thread; // global

void schedule1 () {
    unsigned status = interrupt_disable();
    push_list(ready_list, cur_thread);
    schedule();
    set_interrupt_status(status);
}

void schedule () {
    struct thread *prev = cur_thread;
    struct thread *next = pop_list(ready_list);
    cur_thread = next;
    context_switch(prev, next);
}
```

Write the pseudocodes of `cond_wait` and `cond_signal`. You can use `cur_thread`, `ready_list`, `schedule1`, and `schedule` function directly in your pseudocode. [1]

2.

```
void release(struct lock *l) {
    unsigned status = interrupt_disable();
    struct thread *t = list_pop(l->wait_list);
    if (t)
        list_push(ready_list, t);
    set_interrupt_status(status);
    l->value = 1;
}
```

Is this lock implementation correct? If the answer is no, explain using an example. If the answer is yes, what is the advantage of restoring interrupt status after `l->value` is set to one as discussed in the class. [1]

2 Fork [2 marks]

Execute the following program. `getpid()` returns the `pid` of the current process. `getppid()` returns the `pid` of the parent process.

```
#include <unistd.h>
#include <stdio.h>
#include <sys/wait.h>

int main ()
{
  int pid, i, status;

  printf ("main %d parent %d\n", getpid(), getppid());
  for (i = 0; i < 3; i++) {
    pid = fork ();
    if (pid < 0) {
     printf ("Unable to fork\n");
     return 0;
    }
    if (pid != 0) {
        waitpid (pid, &status, 0);
    }
  }
  printf ("process %d (parent %d) is terminating\n", getpid(), getppid());
  return 0;
}
```

2.1 Turn in:

- The output of the program.
- Draw a tree of the parent-child relationships. A node of the tree contains the `pid` of the process. A directed edge between two nodes represents the parent-child relationship. E.g., 10 -> 11 means process with `pid` 10 is the parent of the process with `pid` 11.

3 Pipes [2 Marks]

The routine below is trying to implement ``‘‘ls | wc -l’’``. You need to use `close` and `dup` system calls to redirect the `STDOUT` of `ls` to the `STDIN` of `wc`. The effect of running your modified code should be same as running the command ``‘‘ls | wc -l’’``. You have to make changes before both the `execv` system calls.

3.1 Turn in:

- What are your changes before the first `execv` (marked as patch-1)?
- What are your changes before the second `execv` (marked as patch-2)?

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

//ls | wc -l

int main()
{
    int pid;
    int fd[2];
    int ret;

    ret = pipe (fd);

    if (ret == -1) {
        printf ("Unable to create pipe\n");
        return 0;
    }
    pid = fork ();
    if (pid == 0) {
        /* Verify that ls exists at /bin/ls */
        /* to verify run: which ls */
        char* const args[] = {"/bin/ls", NULL};

        /* write your code here */
        /* patch-1 */

        ret = execv (args[0], args);
        /* NOT REACHED*/
        printf ("failed to exec ls\n");
    }
    else if (pid > 0) {
        /* Verify that wc exists at /usr/bin/wc */
        /* to verify run: which wc */
        char* const args[] = {"/usr/bin/wc", "-l", NULL};

        /* write your code here */
        /* patch-2 */

        execv (args[0], args);
        /* NOT REACHED*/
        printf ("failed to exec wc\n");
    }
    else {
        printf ("Unable to fork\n");
    }
    return 0;
}
```

How to submit

Submit your handwritten homework in the submission box placed at the old academic building (2nd floor). The box will be placed on days when the homework is due.