Synchronization

```
// Read 3 files, and output their lines in any order

#define NLINES 100000
#define LINESIZE 80
typedef char line_t[LINESIZE];
line_t lines[NLINES];

int main(int argc, char *argv[])
{
    int i;
    int lineno = 0;

    // argv[1..3] name 3 files with lines of 80 characters or less.
    // Read the lines from all three files into the lines[] array,
    // in some arbitrary order.
    // Don't worry about error conditions.

    for (i = 0; i < 3; i++) {
        FILE *f = fopen(argv[i + 1], "r");
        if (f == NULL)
            abort();
        while (fgets(lines[lineno], LINESIZE, f) != NULL)
            lineno++;
        fclose(f);
    }

    for (i = 0; i < lineno; i++)
        fputs(lines[i], stdout);
}
```

```
// Threaded version, no synchronization
int lineno;

void thread_run(void *thunk)
{
    FILE *f = (FILE *) thunk;
    while (fgets(lines[lineno], LINESIZE, f) != NULL)
        lineno++;
    fclose(f);
    pthread_exit(0);
}

int main(int argc, char *argv[])
{
    int i;
    void *thunk;
    pthread_attr_t thread_attr;
    pthread_t threads[3];

    pthread_attr_init(&thread_attr);

    for (i = 0; i < 3; i++) {
        FILE *f = fopen(argv[i + 1], "r");
        if (f == NULL
            || pthread_create(&threads[i], &thread_attr, thread_run, f) < 0)
            abort();
    }

    for (i = 0; i < 3; i++)
        pthread_join(threads[i], &thunk);

    for (i = 0; i < lineno; i++)
        fputs(lines[i], stdout);
}
```

```
// Threaded version, with synchronization
int lineno;

void thread_run(void *thunk)
{
    FILE *f = (FILE *) thunk;
    while (1) {
        int my_lineno;

        acquire_lock();
        my_lineno = lineno;
        lineno++;
        release_lock();

        if (fgets(lines[my_lineno], LINESIZE, f) == NULL) {
            fclose(f);
            pthread_exit((void *) my_lineno);
        }
    }
}

int main(int argc, char *argv[])
{
    int i, empty_linenos[3];
    void *thunk;
    pthread_attr_t thread_attr;   pthread_t threads[3];

    pthread_attr_init(&thread_attr);

    for (i = 0; i < 3; i++) {
        FILE *f = fopen(argv[i + 1], "r");
        if (f == NULL
            || pthread_create(&threads[i], &thread_attr, thread_run, f) < 0)
            abort();
    }

    for (i = 0; i < 3; i++) {
        pthread_join(threads[i], &thunk);
        empty_linenos[i] = (int) thunk;
    }

    // At this point all threads have joined; we are the only thread running.
    // Fill in the "holes".
    for (i = 0; i < 3; i++) {
        memcpy(lines[empty_linenos[i]], lines[lineno - 1], LINESIZE);
        lineno--;
    }

    for (i = 0; i < lineno; i++)
        fputs(lines[i], stdout);
}
```

```
// Non-blocking I/O: event-driven programming

int main(int argc, char *argv[])
{
    int i, lineno = 0;
    fd_set fds;
    FILE *fs[3];

    for (i = 0; i < 3; i++) {
        fs[i] = fopen(argv[i + 1], "r");
        if (fs[i] == NULL)
            abort();
        fcntl(fileno(fs[i]), F_SETFL, O_NONBLOCK);
    }

    while (fs[0] || fs[1] || fs[2]) {
        for (i = 0; i < 3; i++)
            if (fs[i] != NULL) {
                if (fgets(lines[lineno], LINESIZE, fs[i]) != NULL)
                    lineno++;
                else if (errno != EAGAIN) {
                    fclose(fs[i]);
                    fs[i] = NULL;
                }
            }

        // Wait for data to be available on one of the fs
        FD_ZERO(&fds);
        for (i = 0; i < 3; i++)
            if (fs[i] != NULL)
                FD_SET(&fds, fileno(fs[i]));
        select(MAXFD, &fds, NULL, NULL, NULL);
    }

    for (i = 0; i < lineno; i++)
        fputs(lines[i], stdout);
}
```