Starvation

Assuming that jobs keep being submitted by users in a continuous scenario rather than a finite input from a file then starvation will not occur. Observing the job admission stage every new input of a job should be appended to the last index of pending_jobs followed by incrementing the pending_admission counter. Parse_jobs adds the last line of the text file to the head of pending which is problematic if continuous input is added this way. Starvation may occur since older jobs may not execute if the head of the pending list is constantly updating to newer jobs. Admit_jobs does not encounter starvation due to admitting in FIFO ordering and locking after each iteration of the while loop rather than the entire task. Observing the job execution stage, if jobs exist in the admitted_jobs the function will lock the queue such that only one job is processed at a time. Jobs waiting to be completed will wait a finite period of time for all resources to be acquired to allow the processing job to be executed (Line 219). Even if every job required the same resources the same outcome occurs as the resources become locked and only unlocked when the job is completed (Line 226). Hence the executor will process every job regardless of the input being continuous.

A data race

```
==10485== Possible data race during read of size 4 at 0x10C2F4 by thread #4
==10485== Locks held: 2, at addresses 0x10C068 0x10C0B8
==10485==    at 0x109DCC: execute_jobs (jobs.c:225)
==10485==    by 0x4842B1A: ??? (in
/usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==10485==    by 0x489A608: start_thread (pthread_create.c:477)
==10485==    by 0x49D4132: clone (clone.S:95)
```

This valgrind error is telling us that there exists a possible data race error that could be a result of a missing mutex lock. This is important because not having exclusive access to variables could result in incorrect data and result in segmentation faults/bad behaviours.

A missing (and necessary) unlock. Hint: remove a necessary mutex unlock call.

```
==10670== Thread #2: Attempt to re-lock a non-recursive lock I already hold
==10670==    at 0x483FE16: ??? (in
/usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==10670==    by 0x109C81: execute_jobs (jobs.c:205)
==10670==    by 0x4842B1A: ??? (in
/usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==10670==    by 0x489A608: start_thread (pthread_create.c:477)
==10670==    by 0x49D4132: clone (clone.S:95)
==10670==  Lock was previously acquired
==10670==    at 0x4842E11: ??? (in
/usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==10670==    by 0x109C9A: execute_jobs (jobs.c:209)
```

```
==10670==    by 0x4842B1A: ??? (in
/usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==10670==    by 0x489A608: start_thread (pthread_create.c:477)
==10670==    by 0x49D4132: clone (clone.S:95)
```

This valgrind error is telling us that the code is attempting to re-lock something that is already being held/locked. This happens when you forget to unlock a mutex lock. This bug could be a big problem because it could cause starvation for waiting threads and deadlock jobs from executing.

Unlocking a not-locked mutex. Hint: for instance, you could add an extra unlock call on a mutex which is already unlocked.

```
==10809== Thread #4 unlocked a not-locked lock at 0x10C250
==10809==    at 0x48403F6: ??? (in
/usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==10809==    by 0x109EEA: execute_jobs (jobs.c:244)
==10809==    by 0x4842B1A: ??? (in
/usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==10809==    by 0x489A608: start_thread (pthread_create.c:477)
==10809==    by 0x49D4132: clone (clone.S:95)
```

This valgrind error is telling us that we are attempting to unlock a mutex lock that is not locked. This bug could be a problem as it results in undefined behaviour and it may break the code.

Unlock a mutex held by a different thread.

```
==11357== Thread #4 unlocked lock at 0x10C540 currently held by thread #3
==11357==    at 0x48403F6: ??? (in
/usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==11357==    by 0x109EF6: execute_jobs (jobs.c:246)
==11357==    by 0x4842B1A: ??? (in
/usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==11357==    by 0x489A608: start_thread (pthread_create.c:477)
==11357==    by 0x49D4132: clone (clone.S:95)
==11357==  Lock at 0x10C540 was first observed
==11357==    at 0x483FEDF: ??? (in
/usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==11357==    by 0x109B29: admit_jobs (jobs.c:148)
==11357==    by 0x4842B1A: ??? (in
/usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==11357==    by 0x489A608: start_thread (pthread_create.c:477)
==11357==    by 0x49D4132: clone (clone.S:95)
==11357==  Address 0x10c540 is 0 bytes inside data symbol "mp"
```

```
==11357== Thread #9: Bug in libpthread: write lock granted on mutex/rwlock which is currently
wr-held by a different thread
==11357==    at 0x483FEDF: ??? (in
/usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==11357==    by 0x109B29: admit_jobs (jobs.c:148)
==11357==    by 0x4842B1A: ??? (in
/usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==11357==    by 0x489A608: start_thread (pthread_create.c:477)
==11357==    by 0x49D4132: clone (clone.S:95)
```

This valgrind error is telling us that thread 4 unlocked a mutex currently held by thread 3. This error is important because it could result in undefined behaviour where thread 4 could cause data race conditions within thread 3 if it were to unlock the mutex lock prior to thread 3 finishing executing.

Calling pthread_cond_wait with a not-locked mutex, or one locked by a different thread.

```
==11650== Thread #2: pthread_cond_{timed}wait called with mutex held by a different thread
==11650==    at 0x4842BC5: ??? (in
/usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==11650==    by 0x109F01: execute_jobs (jobs.c:246)
==11650==    by 0x4842B1A: ??? (in
/usr/lib/x86_64-linux-gnu/valgrind/vgpreload_helgrind-amd64-linux.so)
==11650==    by 0x489A608: start_thread (pthread_create.c:477)
==11650==    by 0x49D4132: clone (clone.S:95)
```

This valgrind error is telling us that there exists a pthread_cond_wait called with a mutex held by a different thread or with a not-locked mutex. This is an important error to catch because you could miss a wakeup signal if the thread finishes executing before another thread reaches the pthread_cond_wait line.