CSCI-6647 Program 5: Threads/Musical Chairs | Krikor Herlopian/Ricardo Aliwalas | 02-Nov-2020

We used the following checklists to ensure we met the program requirements:

2.0 The Model

- ✓ struct type defined in `Model.hpp` with variables shared by Mom and Kid threads.
- ✓ Mutex lock to control all the other parts of the model
  - We employed locking in Mom.cpp and Kid.cpp
- ✓ int nChairs
- ✓ Pointer to chair array (locked when changing)
- ✓ int nMarching
- ✓ (2) condition variables Kid uses to signal Mom
  - We employ the condition variables in Kid to signal Mom in `Kid::play()`, `Kid::doMarch()`, and `Kid::doSit()`
- ✓ Model constructor to initialize nChairs and allocate array of chairs
  - parameter to constructor - number of charis needed (nKids-1)
- ✓ Matching deconstructor

2.1 Global Function

- ✓ When you create a thread, supply name of global function that the thread will run.
  - Used to start a thread, calls `Kid::play()` in `Global.hpp` .
- ✓ The actual parameter must be a Kid* - see variable k
  - `k -> play();`
- ✓ The call on pthread_create should be in the Kid constructor
- ✓ `const char* sigName(int sig)` that translates the numeric signal codes to strings and returns the string

2.2 The Kid class

- ✓ Private data parts
  - pointer to the shared Model
    - `Model* m;`
  - Kid's ID number and tid:
    - `int id;`
    - `pthread_t tid;`
  - A signal set to define the signals a Kid will listen for
    - `sigset_t set;`
  - subscript of the chair that a Kid will try to capture next
    - `int wantSeat;`
  - subscript of the chair a Kid has captured on this round
    - int seatNumber;

- ✓ Kid Functions
  - Constructor that accepts a Model* and an integer ID number
    - `Kid::Kid(Model* model, int idNumber)`
    - Initialize the signal set to listen for SIGUSR1, SIGUSR2, and SIGQUIT
    - Create a thread for this Kid and store its thread id in the Kid's tid field
      - `pthread_create(&tid, NULL, startThread, (void*) this);`
  - Get functions for the ID number and the tid.
    - `int Kid::getId()`
    - `pthread_t Kid::getTid()`
  - Predicate (boolean function) that returns value indicating whether the Kid sitting/standing
    - `bool isSitting();`

- mutator, `standUp()`, that Mom will call at the beginning of each round (initialize all chairs to -1)
- `doMarch()` Called from play() when the Kid receives the SIGUSR1 signal.
- `doSit()` Called from play() when the Kid receives the SIGUSR2 signal
- `play()` The thread's main function
  - we named this function `Mom::PlayOneRound(Model* m, Kid* players[]`

2.3 Mom – the main function

- ✓ read an integer nKids from the command line
- ✓ Instantiate the model - The argument to the constructor must be the number of chairs needed, which is equal to 1 less than nKids
  - `m = Model(nKids-1);`
- ✓ Create and initialize an array of nKids Kid pointers

```
 Kid* players[nKids];
 for (int i=0; i<nKids; i++) {
     players[i] = new Kid(&m, i);
 }
```

- ✓ In a loop, call the playOneRound() function. The parameters are a pointer to the model and a pointer to the array of Kids

```
for (int x=0; x<(nKids-1); x++) {
    PlayOneRound(&m,players);
    cout << "\nNUMBER OF PLAYERS Left:" <<  stillIn << endl;
}
```

- ✓ When there is only one player left, Mom announces the winner and tells that last Kid to go home.

```
cout << "\nCongratulations Winner is Player: " << players[0]->getId() << " , GO
HOME NOW!" << endl;
```

- ✓ The main function ends with the usual pthread_join loop and appropriate comments. Remember to pthread_exit.

`PlayOneRound( Model* m, Kid* players[] )`

- ✓ Initialize all the chairs in the shared model to the empty state (-1) and the number of kids marching to 0.

```
 for (int i=0; i<m->nChairs; i++) {
     m->chairs[i]=-1;
 }
 m→nMarching=0;
```

- ✓ Initialize a local variable to the number of kids who are still in the game. Mom needs to tell the kids to reinitialize their status (get up out of the chairs). Call each Kid's standUp() function, then send each a USR1 signal.

```
for (int i=0; i<stillIn; i++) {
      players[i]->standUp();
 }
  usleep(100000);
  pthread_mutex_lock(&m→mx);
  for (int i=0; i<stillIn; i++) {
      pthread_kill(players[i]->getTid(), SIGUSR1);
  }
```

- ✓ Wait until every thread has been scheduled and knows it is supposed to be marching.

- ✓ To avoid a busy wait, we will use a loop that tests a condition variable for this communication.
- ✓ To end the music, Mom will send a USR2 signal to each of the kids. Then she must wait until all the kids stop marching. Use a loop and a condition variable, as before.

```
usleep(100000);
pthread_mutex_lock(&m->mx);
for (int i=0; i<stillIn; i++) {
    pthread_kill(players[i]->getTid(), SIGUSR1);
}
```

- ✓ When all the kids have stopped marching, Mom must remove a child and a chair from the game.

```
for (int i=0; i<stillIn; i++) {
    if (!players[i]->isSitting()) {
        found = i;
        cout << players[i]->getId() << "\t\t" << (players[i]->isSitting() ? "Yes" : "No" ) \
            << "\t\t\t" << "KILLED " << players[i]->getSeatNumber() << endl;
        pthread_kill(players[i]->getTid(), SIGQUIT);
        pthread_join(players[i]->getTid(),NULL);
    } else {
        cout << players[i]->getId() << "\t\t"<<(players[i]->isSitting()  ? "Yes" : "No" ) \
            << "\t\t\t" << players[i]->getSeatNumber() << endl;
    }
}
swap(players[found],players[(stillIn-1)]);
```

The Kids

- ✓ Define a function `takeTurn()` for the Kids (threads) to execute. When the first USR signal is received, initialize any local variables to the starting state, download the current value of nKidsChairs, and compute a random chair number between 0 and nKidsChairs − 1. Then wait for the second signal; we will pretend that the kids are marching in a circle during this waiting time. (Note: we called this `play()` in Kid.cpp)
- ✓ When the second signal is received, start with the computed random chair number and test chairs, in sequence, until an empty chair is found. Then lock the chairs and try to store the thread ID in the chair you found.

```
// Called from play() when the Kid receives the SIGUSR1 signal
void Kid::doMarch(){
    pthread_mutex_lock(&m->mx);
    wantSeat = rand() % (m->nChairs);
    m->nMarching++;
    pthread_cond_signal(&m->cv2);   // wake up mom I am marching
    pthread_mutex_unlock(&m->mx);
}
```

- ✓ If you succeed in getting the chair and storing your number in it, wait for the next starting signal. If you fail to get that chair, keep going and hope you can find a different one. In either case, make sure to UNLOCK the chairs.

```
void Kid::doSit(){
    pthread_mutex_lock(&m->mx);
    int size = m->nChairs+1;
    pthread_mutex_unlock(&m->mx);
    int mWantSeat = wantSeat;
    if (seatNumber == -1) {
        for (int i=0; i<size; i++) {
            pthread_mutex_lock(&m->mx);
            if (m->chairs[wantSeat]==-1) {
                m->chairs[wantSeat]= id;
```

```
                    seatNumber = wantSeat;
                    pthread_mutex_unlock(&m->mx);
                    break;
                } else {
                    wantSeat++;
                    // assume we got 6 sits, and we started checking from 4th. We check 4, 5, 6
                    // then we need to check 0,1,2,3..and at 4th we need to stop since we checked
                    // already.
                    if (wantSeat == mWantSeat) {
                        wantSeat = -1;
                        seatNumber = -1;
                    }
                    else if(wantSeat >= m->nChairs) {
                        // since we started checking for seats from random seat number
                        // assume we got 6 sits, and we started checking from 4th. We check
                        // 4, 5, 6
                        // then we need to check 0. Thats why we reset.
                        wantSeat = 0;
                    }
                }
                pthread_mutex_unlock(&m->mx);
            }
            pthread_mutex_lock(&m->mx);
            m->nMarching++;
            pthread_cond_signal(&m->cv);  // wake up mom, I am done sitting
            pthread_mutex_unlock(&m->mx);
        }
    }
```

Testing

We tested our code successfully on the following platforms

```
Intel x86_64  : Debian 10 (4.19.0-12-amd64)
Intel x86_64  : CentOS Linux release 7.8.2003
Intel x86_64  : MacOS 10.15.7
ARM v7l32-bit : Raspbian GNU/Linux 10 (5.4.51-v7l+)
```

We did run into some issues which we were able to solve via system tuning (see Lessons Learned below). Test runs using 3, 5, 500 and 2,000 kids are shown here:

```
$ time ./p5 3

---------------------------------------------------------------
      Krikor Herlopian and Ricardo Aliwalas
      CSCI 6647
      Mon Nov  2 2020    16:26:38
---------------------------------------------------------------

--------------------------NEW ROUND--------------------------
Child        SITTING            seat
-------------------------------------------------------------
2            No                 KILLED -1
0            Yes                0
1            Yes                1

NUMBER OF PLAYERS Left:2

--------------------------NEW ROUND--------------------------
Child        SITTING            seat
```

```
------------------------------------------------------------
0            No                  KILLED -1
1            Yes                 0

NUMBER OF PLAYERS Left:1

Congratulations Winner is Player: 1 , GO HOME NOW!

---------------------------------------------------------------
Normal termination.
./p5 3  0.01s user 0.00s system 0% cpu 1.006 total
```

$ time ./p5 5

```
---------------------------------------------------------------
      Krikor Herlopian and Ricardo Aliwalas
      CSCI 6647
      Mon Nov  2 2020     16:27:26
---------------------------------------------------------------

--------------------------NEW ROUND--------------------------
Child        SITTING             seat
------------------------------------------------------------
3            No                  KILLED -1
4            Yes                 0
2            Yes                 1
1            Yes                 2
0            Yes                 3

NUMBER OF PLAYERS Left:4

--------------------------NEW ROUND--------------------------
Child        SITTING             seat
------------------------------------------------------------
2            No                  KILLED -1
0            Yes                 0
1            Yes                 1
4            Yes                 2

NUMBER OF PLAYERS Left:3

--------------------------NEW ROUND--------------------------
Child        SITTING             seat
------------------------------------------------------------
1            No                  KILLED -1
0            Yes                 0
4            Yes                 1

NUMBER OF PLAYERS Left:2

--------------------------NEW ROUND--------------------------
Child        SITTING             seat
------------------------------------------------------------
0            No                  KILLED -1
4            Yes                 0

NUMBER OF PLAYERS Left:1

Congratulations Winner is Player: 4 , GO HOME NOW!

---------------------------------------------------------------
Normal termination.
./p5 5  0.00s user 0.01s system 0% cpu 2.008 total
```

```
$ time ./p5 500

--------------------------------------------------------------
      Krikor Herlopian and Ricardo Aliwalas
      CSCI 6647
      Mon Nov  2 2020     16:28:35
--------------------------------------------------------------

--------------------------NEW ROUND--------------------------
Child        SITTING             seat
------------------------------------------------------------
465          No                  KILLED -1
229          Yes                 0
258          Yes                 1
471          Yes                 2
269          Yes                 3
443          Yes                 4
391          Yes                 5
183          Yes                 6
460          Yes                 7
136          Yes                 8
. . .
306          Yes                 492
315          Yes                 493
399          Yes                 494
102          Yes                 495
413          Yes                 496
135          Yes                 497
190          Yes                 498

NUMBER OF PLAYERS Left:499
. . .
. . .
. . .
--------------------------NEW ROUND--------------------------
Child        SITTING             seat
------------------------------------------------------------
441          No                  KILLED -1
12           Yes                 0
140          Yes                 1
203          Yes                 2

NUMBER OF PLAYERS Left:3

--------------------------NEW ROUND--------------------------
Child        SITTING             seat
------------------------------------------------------------
12           No                  KILLED -1
203          Yes                 0
140          Yes                 1

NUMBER OF PLAYERS Left:2

--------------------------NEW ROUND--------------------------
Child        SITTING             seat
------------------------------------------------------------
203          No                  KILLED -1
140          Yes                 0

NUMBER OF PLAYERS Left:1

Congratulations Winner is Player: 140 , GO HOME NOW!

--------------------------------------------------------------
Normal termination.
./p5 500 3.37s user 8.19s system 4% cpu 4:14.01 total
```

```
$ time ./p5 2000

---------------------------------------------------------------
       Krikor Herlopian and Ricardo Aliwalas
       CSCI 6647
       Mon Nov  2 2020     16:38:39
---------------------------------------------------------------

--------------------------NEW ROUND--------------------------
Child         SITTING             seat
-------------------------------------------------------------
1790          No                  KILLED -1
689           Yes                 0
868           Yes                 1
344           Yes                 2
18            Yes                 3
347           Yes                 4
1077          Yes                 5
964           Yes                 6
666           Yes                 7
605           Yes                 8

. . .
. . .

--------------------------NEW ROUND--------------------------
Child         SITTING             seat
-------------------------------------------------------------
1557          No                  KILLED -1
1906          Yes                 0
617           Yes                 1
1605          Yes                 2

NUMBER OF PLAYERS Left:3

--------------------------NEW ROUND--------------------------
Child         SITTING             seat
-------------------------------------------------------------
617           No                  KILLED -1
1906          Yes                 0
1605          Yes                 1

NUMBER OF PLAYERS Left:2

--------------------------NEW ROUND--------------------------
Child         SITTING             seat
-------------------------------------------------------------
1906          No                  KILLED -1
1605          Yes                 0

NUMBER OF PLAYERS Left:1

Congratulations Winner is Player: 1605 , GO HOME NOW!

---------------------------------------------------------------
Normal termination.
./p5 2000 45.83s user 92.80s system 13% cpu 17:35.02 total
```

Lessons Learned

We were unable to run a non-trivial amount of kids on the MacOS platform. The problem turned out to be a limit on the number of processes allowed by the user. The limit could be seen via `launchctl limit maxfiles` and can be increased via `sudo launchctl limit maxfiles 65536 200000` .

We were not able to execute with 2,000 kids on our Raspberry Pi. In this case, the number of threads was limited by the amount of memory on the system. The max number of threads can be calculated using this formula:

```
number of threads = total virtual memory / (stack size*1024*1024)
```

where the stack size of 8k is determined by running `ulimit -s` in the shell. On our Pi, with approximately 2gb of available memory we confirmed that we were limited to: 2000000000/(8*1024*1024) = 238 threads at once. Adding swap space can increase the virtual memory and thus the number of threads permitted on a system with limited available memory.

The operating system itself will have an outer limit for number of threads allowed. This limit is found here:

```
$ cat /proc/sys/kernel/threads-max
126377
```

On CentOS/RedHat/Fedora Linux, the maximum number of processes allowed for a *user* can be configured via the file `/etc/security/limits.d/20-nproc.conf` . On Debian linux, the max number of processes can be modified via `/etc/security/limits.conf` .

To observe the number of threads executed by the program in real time, we used the ps command with the -L option:

```
$ ps -eLf | grep p5 | wc -l
```

We also found that strace could be used to observe system calls in progress. For instance:

```
$ strace -f ./p5
. . .
[pid 24708] write(1, "NUMBER OF PLAYERS Left:2\n", 25NUMBER OF PLAYERS Left:2
) = 25
[pid 24708] write(1, "\n", 1
)           = 1
[pid 24708] write(1, "-------------------------NEW RO"...,
63-------------------------NEW ROUND---------------------------
) = 63
[pid 24708] nanosleep({tv_sec=0, tv_nsec=100000000}, NULL) = 0
[pid 24708] getpid()                    = 24708
[pid 24708] tgkill(24708, 24710, SIGUSR1) = 0
[pid 24710] <... rt_sigtimedwait resumed> {si_signo=SIGUSR1, si_code=SI_TKILL,
si_pid=24708, si_uid=1000}, NULL, 8) = 10 (SIGUSR1)
```

In short, it was extremely valuable to not only compile our code on multiple platforms but also to run it.