# COMP 7500/7506 Lecture 19: Project 3-8 A Sample AUbatch

typedef unsigned int u\_int;

//#define LOW\_ARRIVAL\_RATE /\* Long arrivel-time interval \*/

#define LOW\_SERVICE\_RATE /\* Long service time \*/

#define CMD\_BUF\_SIZE 10 /\* The size of the command queueu \*/

#define NUM\_OF\_CMD 5 /\* The number of submitted jobs \*/

#define MAX\_CMD\_LEN 512 /\* The longest commandline length \*/

void \*commandline( void \*ptr ); /\* Job submissions and scheduling \*/

void \*executor( void \*ptr ); /\* Job execution \*/

pthread\_mutex\_t cmd\_queue\_lock; /\* Lock for critical sections \*/

pthread\_cond\_t cmd\_buf\_not\_full; /\*Condition var for buf\_not\_full\*/

pthread\_cond\_t cmd\_buf\_not\_empty; /\* CV for buf\_not\_empty \*/

/\* Global shared variables \*/

u\_int buf\_head;

u\_int buf\_tail;

u\_int count;

char \*cmd\_buffer[CMD\_BUF\_SIZE];

int main() {

pthread\_t command\_thread, executor\_thread; /\* Two concurrent threads \*/

char \*message1 = "Command Thread";

char \*message2 = "Executor Thread";

int iret1, iret2;

/\* Initilize count, two buffer pionters \*/

count = 0;

buf\_head = 0;

buf\_tail = 0;

/\* Create two independent threads:command and executors \*/

iret1 = pthread\_create(&command\_thread, NULL, commandline, (void\*) message1);

**iret2 = pthread\_create(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_);**

/\* Initialize the lock the two condition variables \*/

pthread\_mutex\_init(&cmd\_queue\_lock, NULL);

pthread\_cond\_init(&cmd\_buf\_not\_full, NULL);

**pthread\_cond\_init( \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_);**

/\* Wait till threads are complete before main continues. Unless we

wait we run the risk of executing an exit which will terminate

the process and all threads before the threads have completed.\*/

pthread\_join(command\_thread, NULL);

**pthread\_join(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_);**

printf("command\_thread returns: %d\n",iret1);

printf("executor\_thread returns: %d\n",iret1);

exit(0);

}

/\*

\* This function simulates a terminal where users may

\* submit jobs into a batch processing queue.

\* Note: The input parameter (i.e., \*ptr) is optional.

\* If you intend to create a thread from a function

\* with input parameters, please follow this example.

\*/

void \*commandline(void \*ptr) {

char \*message;

char \*temp\_cmd;

u\_int i;

char num\_str[8];

size\_t command\_size;

message = (char \*) ptr;

printf("%s \n", message);

/\* Enter multiple commands in the queue to be scheduled \*/

for (i = 0; i < NUM\_OF\_CMD; i++) {

/\* lock the shared command queue \*/

pthread\_mutex\_lock(&cmd\_queue\_lock);

printf("In commandline: count = %d\n", count);

while (count == CMD\_BUF\_SIZE) {

pthread\_cond\_wait**( \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)**;

}

/\* Dynamically create a buffer slot to hold a commandline \*/

#ifdef STATIC\_COMMAND

cmd\_buffer[buf\_head] = malloc(strlen("process -help -time ") + 1);

strcpy(cmd\_buffer[buf\_head], "./process -help -time ");

sprintf(num\_str, "%d", i);

strcat(cmd\_buffer[buf\_head], num\_str);

#else

pthread\_mutex\_unlock(&cmd\_queue\_lock);

printf("Please submit a batch processing job:\n");

printf(">");

temp\_cmd = malloc(MAX\_CMD\_LEN\*sizeof(char));

getline(&temp\_cmd, &command\_size, stdin);

**pthread\_mutex\_lock( \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_);**

cmd\_buffer[buf\_head]= **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;**

#endif

printf("In commandline: cmd\_buffer[%d] = %s\n", buf\_head, cmd\_buffer[buf\_head]);

count++;

/\* Move buf\_head forward, this is a circular queue \*/

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;**

if (buf\_head == CMD\_BUF\_SIZE)

buf\_head = 0;

pthread\_cond\_signal**( \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)**;

/\* Unlok the shared command queue \*/

pthread\_mutex\_unlock(&cmd\_queue\_lock);

#ifdef LOW\_ARRIVAL\_RATE /\* Simulate a low arrival rate \*/

sleep(2); /\* Simulate an arrival time of 2 seconds \*/

#endif

} /\* end for \*/

}

/\*

\* This function simulates a server running jobs in a batch mode.

\* Note: The input parameter (i.e., \*ptr) is optional.

\* If you intend to create a thread from a function

\* with input parameters, please follow this example.

\*/

void \*executor(void \*ptr) {

char \*message;

u\_int i;

message = (char \*) ptr;

printf("%s \n", message);

for (i = 0; i < NUM\_OF\_CMD; i++) {

/\* lock and unlock for the shared process queue \*/

pthread\_mutex\_lock(&cmd\_queue\_lock);

printf("In executor: count = %d\n", count);

while (count == 0) {

pthread\_cond\_wait**( \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)**;

}

/\* Run the command scheduled in the queue \*/

count--;

printf("In executor: cmd\_buffer[%d] = %s\n", buf\_tail, cmd\_buffer[buf\_tail]);

/\*

\* Note: system() function is a simple example.

\* You should use execv() rather than system() here.

\*/

system(cmd\_buffer[buf\_tail]);

/\* Free the dynamically allocated memory for the buffer \*/

free(cmd\_buffer[buf\_tail]);

#ifdef LOW\_SERVICE\_RATE

sleep(2); /\* Simulate service time of 2 seconds \*/

#endif

/\* Move buf\_tail forward, this is a circular queue \*/

buf\_tail++;

if (buf\_tail == CMD\_BUF\_SIZE)

buf\_tail = 0;

pthread\_cond\_signal**( \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)**;

/\* Unlok the shared command queue \*/

pthread\_mutex\_unlock(&cmd\_queue\_lock);

} /\* end for \*/

}