Parallel Matrix Multiplication Sum (pmms)
OPERATING SYSTEMS ASSIGNMENT

GARLAND, Christopher (15560955)

September 24, 2016

1 ReadMe

This program will calculate the product of two matrices, and sum the product matrix in parallel by using multiple processes/threads. Test files have been provided, and will be discussed further in section 3.0.

There is an emphasis on process/thread creation & synchronization (section 2.0) and less on error handling. This means that there are several conditions under which the program will not run — discussed in section 3.0.

This project should compile and run on any of the lab machines. `gdb` was used to test the program. Where possible `valgrind` has been used to check for memory leaks. Further discussion of testing in section 3.0.

1.1 Instructions for Compilation

- Compile all binary files and report: `make`
- Compile only binary files: `make bin`
- Compile only report: `make report`
- Delete all generated files: `make clean`

1.2 Instructions for Execution

Included within this project are eight matrix description files that can be provided as arguments to the programs. The files (along with their dimensions) are to be used as program arguments in the following 4 pairs.

1. `description-A description-B 3 2 4`
2. `description-C description-D 3 3 3`
3. `description-E description-F 3 5 4`
4. `description-G description-H 5 2 5`

For the purpose of these instructions, pair 1 will be used. The programs can be executed as follows.

- Processes: `./pmms-process description-A description-B 3 2 4`
- Threads: `./pmms-thread description-A description-B 3 2 4`
- Both: `./pmms description-A description-B 3 2 4`

2 Mutual Exclusion & Synchronization

Mutual exclusion and process/thread synchronization is achieved with the use of mutex locks and semaphores. In both cases (processes & threads), we are faced with the producer/consumer problem and have a bounded buffer with a size of 1.

2.1 Using Processes

The main strategies to achieve process synchronization were to use shared memory blocks and semaphores. Source can be found in `process-control.c`

A semaphore struct was created and used. The source code can be viewed in the file `semaphore.h`. The struct contains a `mutex` variable that is used as a binary lock, and is initialized to 0. There are two other variables `empty` and `full`. As the buffer is only of size 1, these variables are binary in nature, and are initialized to 1 and 0 respectively.

The tactic employed is further illustrated in Figure 1.

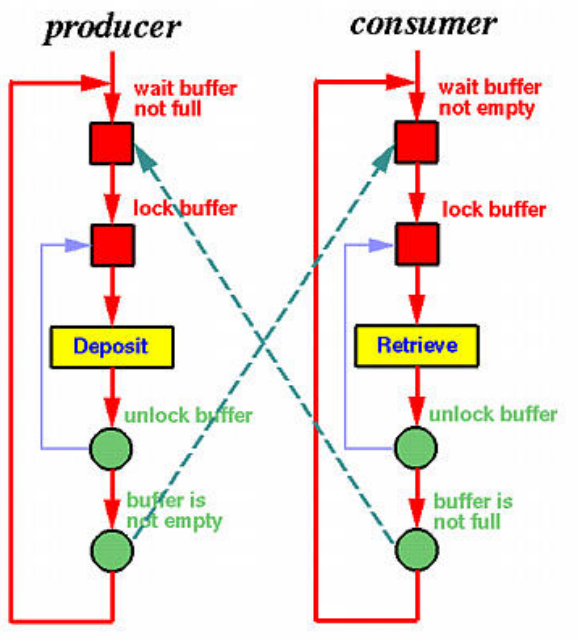

Figure 1: Producer/Consumer with bounded buffer.

2.2 Using Threads

Mutex lock with condition variables were used to achieve mutual exclusion. No shared memory blocks were created as threads already share memory with the parent.

Source code can be found in `thread-control.c`

3 Known Faults & Testing

The focus for this assignment was on process/thread creation and synchronization and less on error handling. This means there are conditions upon which this program will not run.

3.1 Input Files & Error Handling

The program truncates the first 2 lines of a file before reading in the matrix values. This is by no means the best way to implement a file read but as it stands, there are very strict rules for input files...

- Files must have exactly two lines (of anything) before the matrix values
- The number of matrix values in the files MUST match the dimensions provided.

3.2 Memory Leaks

There are known memory leaks. Here is a `valgrind` leak check:

```
==30966==
==30966== HEAP SUMMARY:
==30966==     in use at exit: 1,789 bytes in 27 blocks
==30966==   total heap usage: 29 allocs, 2 frees, 2,301 bytes allocated
==30966==
==30966== LEAK SUMMARY:
==30966==    definitely lost: 788 bytes in 5 blocks
==30966==    indirectly lost: 0 bytes in 0 blocks
==30966==      possibly lost: 0 bytes in 0 blocks
==30966==    still reachable: 1,001 bytes in 22 blocks
==30966==         suppressed: 0 bytes in 0 blocks
==30966== Rerun with --leak-check=full to see details of leaked memory
==30966==
==30966== For counts of detected and suppressed errors, rerun with: -v
==30966== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 14 from 8)
==30967==
==30967== HEAP SUMMARY:
==30967==     in use at exit: 1,789 bytes in 27 blocks
==30967==   total heap usage: 29 allocs, 2 frees, 2,301 bytes allocated
==30967==
==30967== LEAK SUMMARY:
==30967==    definitely lost: 788 bytes in 5 blocks
==30967==    indirectly lost: 0 bytes in 0 blocks
==30967==      possibly lost: 0 bytes in 0 blocks
==30967==    still reachable: 1,001 bytes in 22 blocks
==30967==         suppressed: 0 bytes in 0 blocks
==30967== Rerun with --leak-check=full to see details of leaked memory
==30967==
==30967== For counts of detected and suppressed errors, rerun with: -v
==30967== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 14 from 8)
Subtotal produced by thread with ID 81902448: 62
Subtotal produced by thread with ID 92392304: 134
Subtotal produced by thread with ID 102882160: 206
Total: 402
```

4 Example I/O

In addition to the required output, this program outputs a system call to `ps` to a file called `psout`. This is to prove the existence of the running processes. Here is an example of `psout` after running the program...

```
  PID TTY           TIME CMD
 1089 ttys001    0:00.67 -zsh
 2896 ttys001    0:00.00 ./pmms description-A description-B 3 2 4
 2897 ttys001    0:00.00 (pmms)
 2898 ttys001    0:00.00 ./pmms description-A description-B 3 2 4
 2899 ttys001    0:00.00 ./pmms description-A description-B 3 2 4
 2900 ttys001    0:00.00 sh -c ps > psout
```

We can clearly see that there were 4 processes created. This is the expected result given the following test input.

4.1 Input

Example of first input file:

```
# Matrix A: 3 x 2

1 2
3 4
5 6
# EOF
```

Example of second input file:

```
# Matrix B: 2 x 4

1 2 3 4
5 6 7 8

# EOF
```

4.2 Output

When run with `./pmms-process description-A description-B 3 2 4`

```
Subtotal produced by process with ID 2897: 62
Subtotal produced by process with ID 2898: 134
Subtotal produced by process with ID 2899: 206
Total: 402
```

When run with `./pmms-thread description-A description-B 3 2 4`

```
Subtotal produced by thread with ID 87359488: 62
Subtotal produced by thread with ID 87896064: 134
Subtotal produced by thread with ID 88432640: 206
Total: 402
```

When run with `./pmms description-A description-B 3 2 4`

```
Subtotal produced by process with ID 2897: 62
Subtotal produced by process with ID 2898: 134
Subtotal produced by process with ID 2899: 206
Total: 402
```

```
Subtotal produced by thread with ID 87359488: 62
Subtotal produced by thread with ID 87896064: 134
Subtotal produced by thread with ID 88432640: 206
Total: 402
```