情報科学演習C

第3回レポート課題

【担当教員】　内山　彰　教員

【提出者】　内藤 圭吾　(09B14049)
ソフトウェア科学コース・3年
u434666c@ics.osaka-u.ac.jp
【提出日】　2016年7月7日

1　課題 3-1

1.1　課題内容

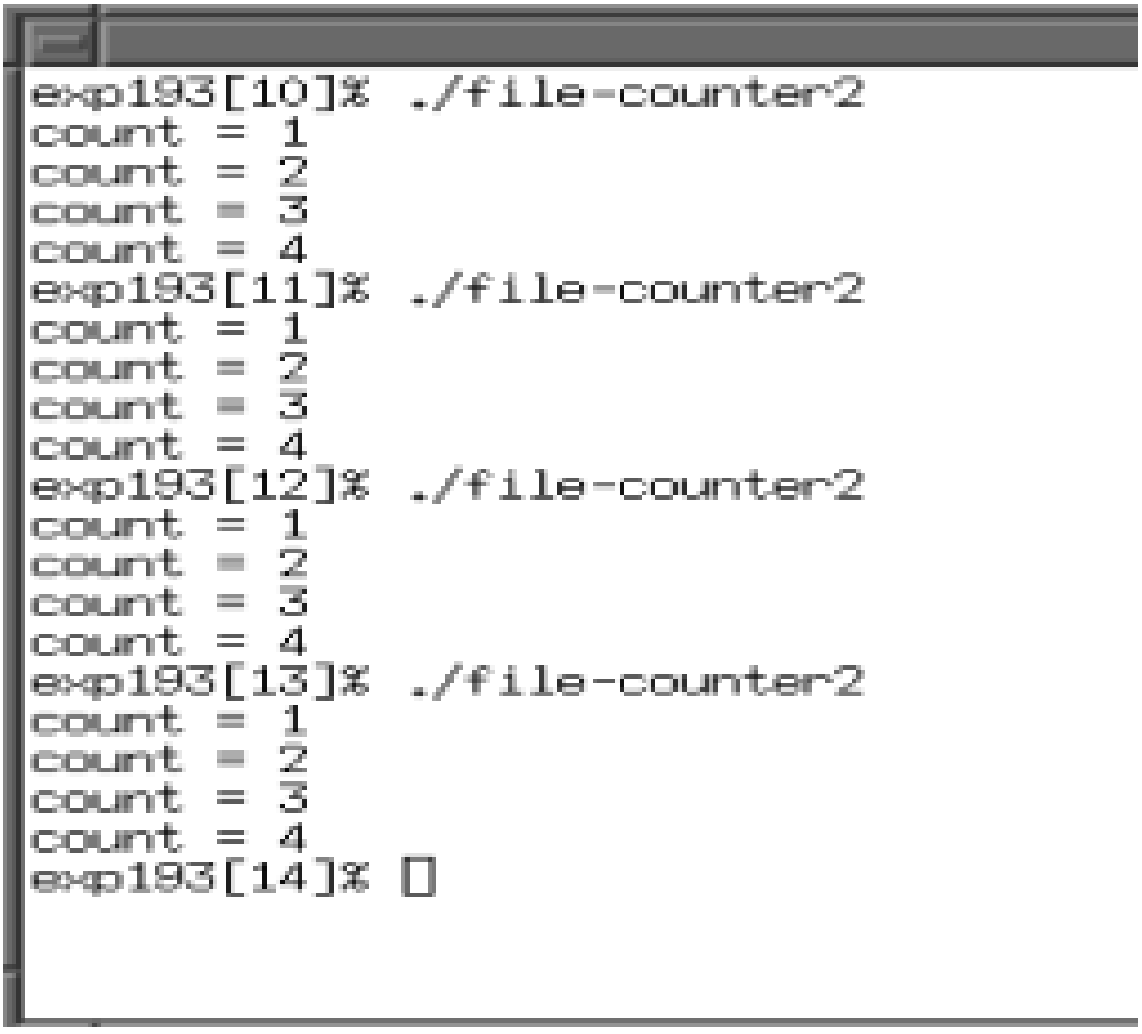
セマフォを用いて排他制御を実現し、与えられたプログラム file-counter.c が確実に正しく動作するように改良する。正しい動作とは、以下の 3 条件を満たす動作である。

- counter ファイルの内容が最終的に 4 になる。
- 何度実行しても同じ結果になる。
- file-counter プログラム単体で動作する。

1.2　プログラムの仕様

引数はなし。制限事項はなし。

1.3　実行結果



1.4 考察

今回は排他制御をセマフォを用いて実装したが、今回の課題では、ファイルのロック機能を用いることで、排他制御を行い、課題を実装することもできる。ファイルのロックは¡sys/file.h¿ライブラリの、flock 関数で行うことができる。flock 関数は第一引数にファイルディスクリプタ、第二引数にどの操作を行うかを表す operation を取る。flock 関数が行うことのできる操作は以下の 3 つがある。

- LOCK_SH : 共有ロックを適用する。
- LOCK_EX : 排他ロックを適用する。
- LOCK_UN : このプロセスの保持している既存のロックを解除する。

共有ロックは、他のプロセスからそのファイルを参照することはできるが、変更することはできない。排他ロックは、他のプロセスからはファイルの参照も変更もできない。今回の場合、排他ロックを行うことで、セマフォを用いた場合と同様の動作をすることができる。

2 課題 3-2-1

2.1 課題内容

pipe.c を拡張し、双方向双方向パイプのプログラム two-way-pipe.c を作成する。

2.2 プログラムの仕様

第一引数に子プロセスから親プロセスへのメッセージ、第二引数に親プロセスから子プロセスへのメッセージを指定する。メッセージは 256 文字以下に制限している。

2.3 実行結果

```
exp193[16]% ./two-way-pipe Hello Hogehoge
Message from parent process:
        Hogehoge
Message from child process:
        Hello
```

2.4 考察

今回メッセージの文字数が 256 文字以下に制限されているが、これは入力された文字列を格納する配列 buf のサイズが 256 であることに起因している。したがって、buf のサイズを変更するだけで、文字数の制限をあげることができる。しかし、後述するが pipe のバッファには 8192bit が上限と言う制限があるので、文字数は文字列送信アルゴリズムを課題 3-2-2 後半の工夫を施さなければ、8192 文字以上の送信はできない。

3　課題 3-2-2

3.1　課題内容

mergesort.c を拡張し、並列版のマージソートを行うプログラムを作成する。

3.2　プログラムの仕様

引数なしで実行すると、ランダムに生成された数値がソートされて標準出力に表示される。

3.3　実行結果

```
exp193[29]% ./mergesort2
Done with sort.
18722997
33602717
139160310
170457686
184716611
190301860
209175999
257655393
268510490
271274882
475268890
532331298
548240959
800227443
875534371
908994462
963980291
992679889
1000134675
1079545348
1087174605
1102370201
1144814923
1194562344
1236476929
1352567843
1352734365
1377141211
1402152006
1418227968
1493350612
1580969089
1598434194
1630723117
1880053793
1976830786
2010575041
2120165911
2137217446
2144602219
```

図 1: 要素数 40 での実行結果

```
Done with sort.
92603
125183
175952
269826
382044
436263
443881
471143
486062
568882
590043
590236
598755
725318
740571
742682
766528
865153
927550
939745
964001
965209
1029533
1038769
1157008
1157554
1241263
1308033
1342499
1347516
1439319
```

図 2: 要素数 40000 での実行結果 (最小値付近)

```
2145064311
2145068290
2145245958
2145320485
2145410344
2145440887
2145485365
2145524072
2145618631
2145634667
2145713096
2145760004
2145788847
2145902095
2146034285
2146260555
2146487687
2146509564
2146576848
2146591871
2146592296
2146630636
2146692280
2146749701
2146950131
2146976073
2146984796
2147060328
2147252887
2147289779
2147294672
2147463094
```

図 3: 要素数 40000 での実行結果 (最大値付近)

3.4　考察

今回、pipe で一度にマージソートした数値を親プロセスに送信していた場合、配列サイズが大きくなった場合、プログラムが途中で停止し、ソートを完了することができない。これは、pipe で送信する際、pipe に渡した値は一度バッファに溜められ、read 関数で、そのバッファから読み出される。この pipe のバッファは 8192byte であり、それ以上のサイズをバッファに溜めようとすると、バッファがあふれ、read 関数でバッファに空きができるまで、動作がブロックされる事が原因である。なので、あまりに大きなサイズのデータを一度に pipe に渡すと処理できずに動作がとまってしまう。

4　課題 3-3-1

4.1　課題内容

alarm 関数を使用せずに、同等の機能を実現する関数 myalarm をマルチプロセスを用いて実装する。

4.2　プログラムの仕様

最後の入力から 10 秒経過でプログラムが終了する。制限はなし。

4.3　実行結果

```
exp193[54]% ./myalarm
hogehoge
echo: hogehoge
hello
echo: hello
foo
echo: foo
This program is timeout.
exp193[55]% ps
  PID TT  STAT      TIME COMMAND
  928  1  Is+    0:00.04 -csh (csh)
  947  2  Ss     0:00.11 csh
 1226  2  R+     0:00.00 ps
```

4.4　考察

今回、以前の myalarm 関数の呼び出しで作られた子プロセスを殺すために static 変数を用いた。これにより、以前作られた子プロセスのプロセス ID を格納し、次以降の呼び出しで殺すことができた。これ以外の実現方法としては、子プロセスが終了や一時停止した際に発生するシグナル、SIGCHLD を用いるものが考えられる。

5　課題 3-3-2

5.1　課題内容

myalarm 関数を利用して、課題 2 で作成した simple-talk-client.c にタイムアウト機能を実装する。この機能は以下の仕様を満たすこと。

- 一定時間入出力が無い場合、メッセージを出力してプログラムを終了する。
- シグナルハンドラ内では、非同期安全な関数のみを利用する。
- プログラムを終了する前に、ソケットを閉じるなど、必要な後処理をした後プログラムを終了すること。

5.2 プログラムの仕様

上記の仕様を満たしている。引数は接続先の端末名。

5.3 実行結果

```
exp205[5]% ./simple-talk-server
Please input your name!
hoge
Thanks you!
[exp193.exp.ics.es.osaka-u.ac.jp]
[foo]:hello
[foo]:hoge
foo
closed
```

```
exp193[68]% ./simple-talk-client exp205
Please input your name!
foo
Thanks you!
hello
hoge
[hoge]:foo
process is timeout
exp193[69]%
```

5.4 考察

今回シグナルハンドラ内で、非同期安全な関数のみを用いる、と言う制限があったが、この理由は、シグナルハンドラ内で非同期安全でない関数を呼び出すと、脆弱性の原因となるからである。そもそも、非同期安全な関数とは、割り込まれている操作を妨げないことが保障されている関数である。つまり、非同期安全でない関数が、割り込み処理を記述するシグナルハンドラに用いられていた場合、割り込み処理を行うことで、もとの操作が損なわれる可能性がある。元の操作が損なわれる例として、課題 3-3-1 でシグナルハンドラ内で用いられていた printf 関数の場合を考える。元の main 関数で printf 関数を呼び出している時に、シグナルが発生し、そのシグナルハンドラ内にも printf 関数が書かれていたとする。その場合、2 つの printf の出力が混ざり合ってしまう。この問題を解決する方法として、排他制御が考えられる。しかし、この場合排他制御を行おうとすると、デッドロックが発生してしまう。そのため、割り込み処理を記述するシグナルハンドラ内には、元の処理の操作を妨げないことが保障されている非同期安全な関数のみを用いることが必要である。そこで、今回は非同期安全でない printf 関数の代わりに、非同期安全である write 関数で置き換えることで対処した。

6 感想

他の講義でも、セマフォを用いた排他制御は学んでいたが、実際にその実装を行ってみるとセマフォに対する理解がよりいっそう深まった。また、前回の課題も含めて、マルチプロセスプログラミングへの理解も少しずつ進んだ。今回はシグナルに対応した動作を変更する課題があったが、これは様々な応用方法があるように感じた。今回も大変身になる課題であった。

参考文献

[1] Oracle マニュアル https://docs.oracle.com/cd/E19683-01/816-3976/gen-26/