平成13年度オペレーティングシステム試験

2002年2月5日

問題は3問，4ページある．

1

今日の典型的なPC/ワークステーション用のオペレーティングシステムが提供する「保護機構」について，その基本的な仕組みを以下の場合について論ぜよ．

1. ディスクやネットワークなどの入出力装置への，ユーザプロセスからの直接アクセスを禁ずる仕組み
2. ユーザプロセスが他のユーザプロセスのメモリを参照・破壊するのを禁ずる仕組み
3. ユーザプロセスが，CPUを独占利用するのを禁ずる仕組み

いずれの場合も，

- ハードウェア (CPU) によって提供される機能
- ソフトウェア (オペレーティングシステム) がそれをどのように利用しているか

を区別しながら簡潔に述べよ．

2

ワトスンは悩んでいた．ワトスンは，オペレーティングシステムが提供する通信プリミティブを使って，同一のコンピュータ上で通信する二つのプログラムPとCを走らせていた．その主要部分のコードは以下のようになっていた．

次ページへ続く

```
P:
P0:  char data[DATA_SZ]; ack[1];
     ...
P1:  for (i = 0; i < 100; i++) {
       /* データを，配列 data に書き込む */
P2:    produce_data(data);
       /* data の中身を C に送信する */
P3:    send(s, data, DATA_SZ, 0);
       /* C4 で送られる返答を待つ */
P4:    recv(s, ack, 1, 0);
P5:  }

C:  (P とは別のプログラム)
C0:  char data[DATA_SZ]; ack[1];
     ...
C1:  for (i = 0; i < 100; i++) {
       /* P3 で送られたデータを配列 data に受け取る */
C2:    recv(s, data, DATA_SZ, 0);
       /* 受け取ったデータを処理 (消費) する */
C3:    consume_data(data);
       /* P に終了を告げる */
C4:    send(s, ack, 1, 0);
C5:  }
```

各行の意図はコメントのとおりである．すなわちこのプログラム P と C を実行すると,「P が produce_data を使って計算した結果を C が受け取り consume_data によって処理し，P に終了通知を返す」という処理が，100 回繰り返される．

このプログラムはそこそこ順調に動いていたが，貪欲なワトスンはある日,「P と C はどうせ同じコンピュータで走るのだから，スレッドと共有メモリを用いて書き直せば今よりずっと速くなるに違いない」と思うに至った．しかし，期待した結果は得られなかった．以下はワトスンがこの謎解きをする際に発した独り言である．君は名探偵ホームズとなってワトスン君が何を勘違いしているのかを教えてあげてほしい．

ワトスン：(なんということだ．私のコンピュータの知識によれば，スレッドと共有メモリを用いたプログラム同士の通信は，わざわざソケット (上記の send/recv) を呼ばねばならない通信方式よりもよりも速いはずなのに．結果は，ええっと，もともとのプログラムが 561ms なのに，新しいプログラムでは 3965 ms もかかっている．)

ワトスンは以下の新しいコードを眺めながら頭をかきむしっている．

次ページへ続く

```
     /* 共有メモリ */
 Q1: volatile int n_in = 0;
 Q2: volatile int n_out = 0;
 Q3: volatile char data[DATA_SZ];
     /* producer スレッド */
 Q4: producer() {
 Q5:   for (i = 0; i < 100; i++) {
         /* consumer が Q15 を実行するまで待つ */
 Q6:     while (n_in > n_out) { /* なにもしない */ }
         /* データを共有メモリ data に書く */
 Q7:     produce_data(data);
         /* 書いたことを知らせる (Q13 を参照) */
 Q8:     n_in++;
 Q9:   }
Q10: }
     /* consumer スレッド*/
Q11: consumer() {
Q12:   for (i = 0; i < 100; i++) {
         /* producer が Q8 を実行するまで待つ */
Q13:     while (n_in == n_out) { /* なにもしない */ }
         /* 共有メモリ data からデータを読む */
Q14:     consume_data(data);
         /* 処理の終了を知らせる (Q6 を参照) */
Q15:     n_out++;
Q16:  }
Q17: }
```

ホームズはワトスンを背にしながらタバコをふかしていた．そしてしばらくしてつぶやいた．

ホームズ「教科書の 164 ページを読みたまえ」

ワトスン「なんだって!? ホームズ，君は僕のコードを一行も見ていないじゃないか」

ホームズ「または君のプログラムを君の安いラップトップではなく，私のサーバで動かしたまえ」

ワトスン「し，しかしサーバといっても CPU は同じ Pentium，クロックがわずかに違うだけ．速くなるといってもせいぜい数十パーセントの違いじゃないだろうか?」

ホームズ「その二つの機械にはもっと決定的な違いがあるのだよ．おそらく私のサーバで実行すれば，10 倍以上は速くなるだろう」

ワトスン「...さっぱりわからないな，ホームズ，いつも君はそうやって...」

ホームズの変わりにあわれなワトスン君に事情を説明せよ．

- なぜ書き換えたプログラムは，元のプログラムよりも圧倒的に遅いのか?
- なぜ書き換えたプログラムは，サーバマシンで実行すれば 10 倍以上速くなるのか? このサーバは，ラップトップと何が「決定的に」違ったのかを予想してみよ．

3

(1) 以下のプログラムを，N の値を様々に変えて実行することを考える．ただし，PAGESIZE はページの大きさである．

```
char a[N * PAGESIZE];

while (1) {
  for (i = 0; i < N; i++) {
     a[i * PAGESIZE]++;
  }
}
```

オペレーティングシステムが正確な LRU ページ置換アルゴリズム (つまり，ページ置き換えの際に，最後に使われたのが最も遠い過去であるようなページを置き換える) を用いているとする．この計算機が持つ物理ページ数を P とする．N を 1 から P を超えて十分大きくした際の，このプログラムのページフォルト率 (1 メモリアクセスがページフォルトを起こす確率) をグラフに描け (横軸 N, 縦軸がページフォルト率)．結果だけでなく，そうなる理由の説明も書くこと．

ただし簡単のため，メモリアクセスは配列 a への参照・更新についてのみ行われるものとする．つまり，このプログラムのそれ以外のメモリ参照はなく，他のプロセスやオペレーティングシステムによるメモリ参照も無視するものとする．

(2) 同じことを，ランダムページ置換 (ページ置き換えの際に置き換えるページを，一様乱数により選択する) について行え．

(3) このプログラムに対しては明らかに LRU はランダムページ置換に劣っている．にもかかわらず，多くのオペレーティングシステムで，LRU を近似するようなアルゴリズムが用いられている．これはどのような考えに基づくものか論ぜよ．