2024 年度オペレーティングシステム期末試験

2025 年 1 月 29 日 (水)

- **問題は 3 問.**
- **この冊子は，表紙が 1 ページ (このページ), 問題が 2-11 ページからなる.**
- 解答用紙の Submit は何度行っても良く, 最後に Submit したものだけが受け付けられるので, **試験終了や, その間際になるのを待たずに** 各自のタイミングで行うこと.
- 試験終了後に Submit するのは**不正行為**となる (Submit はできてしまうが, Submit した時刻が記録されており, 後からそれが発覚することになるので, 決して行わないよう注意せよ).

問題 1

以下の問いに答えよ.

(1) オペレーティングシステムがアプリケーションから保護される仕組みについて説明した文として, 以下のうちから最も的確なものを選べ.

(a) 各プロセスごとに一つの論理アドレス空間を割り当て, アドレス変換機構(MMU)を用いて異なるプロセスが同一の物理ページを用いないようにする.
(b) ファイルへのアクセスやメモリの割当などのためのシステムコール中で適切にアクセス権限を検査することで, 合法でないアクセスを禁止する.
(c) プロセスはユーザーモードで動作させ, オペレーティングシステムはスーパーバイザモードで動作させる.
(d) 仮想化技術を用いて複数の環境を隔離することで, プロセスがオペレーティングシステムのデータを読んだり書いたりできないようにする.

(2) グラフィカルユーザインタフェース (GUI) 環境で, あるアプリケーションの 1 スレッドがシステムコールを一切呼び出さない無限ループに陥ってもマウスカーソルは動くしそのアプリケーションを終了させるための操作が可能である仕組みについて説明した文として, 以下のうちから最も的確なものを選べ.

(a) マウスは CPU で動いているのではないので CPU がそのアプリケーションに占有されても動き続ける.
(b) CPU には複数のコアが存在しており, 1 つのスレッドはその中の 1 つのコアを占有するだけであり, マウスは別のコアで動き続ける.
(c) CPU に対するタイマー割り込みが発生して OS に制御が渡るようになっている.

(3) グラフィカルユーザインタフェース (GUI) 環境では, 数百を超えるプロセスが存在していることも通常である. そのような状態の PC (ノート PC で OS は Linux とする) で, 長時間の計算を行うプログラムを実行したところ, ほぼ CPU の限界性能に近い性能で動いた. 他に数百を超えるプロセスが存在しているにも関わらずそうなる理由として最も的確なものを選べ.

(a) Linux のスケジューラが vruntime という変数をスレッドごとに割当て, CPU 時間を公平に割り当てている.
(b) ほとんどのプロセス (の中のほとんどのスレッド) が中断している.
(c) 長時間の計算を行っているスレッドには CPU 時間を多く割り当てるような優先度付けが行われる.
(d) ノート PC であっても複数のコアを持つマルチコア CPU を搭載している.

(4) 以下の (O1) - (O5) の 5 つのプログラムはいずれも起動すると, 子スレッドを 1 つ作り, 親, 子スレッドそれぞれが関数 `foo` を呼び出し, 子スレッドが終了したあとで, 親スレッドがあるメモリ領域 (変数や, 動的に割り当てられた領域) の値を, `x = ...` または `*p = ...` のように表示するものである. (P1) - (P4) の 4 つのプログラムは同様のことをプロセスを用いて行う. これらのプログラムの出力 (上記の … の部分) について正しく記述しているものが以下のどれであるかを, すべてのプログラムについて(a)-(e)の記号で答えよ.

(a) 常に 0.
(b) 常に 1.

(c) 常に 2.
(d) 常に一定だが, 0, 1, 2 のどれでもない.
(e) 不定 (実行のタイミングによって結果が異なる).

注: 以下のプログラムは文法的には正しいが, `#include` 句が欠けているなどでそのままではコンパイルできないこともある.

(O1)

```
int x = 0;
void foo() {
  x++;
}
int main() {
#pragma omp parallel num_threads(2)
  foo();
  printf("x = %d\n", x);
  return 0;
}
```

(O2)

```
int x = 0;
void foo(int y) {
  y++;
}
int main() {
#pragma omp parallel num_threads(2)
  foo(x);
  printf("x = %d\n", x);
  return 0;
}
```

(O3)

```
void foo(int * p) {
  *p += 1;
}
int main() {
  int * p = (int *)calloc(1, sizeof(int));
  if (!p) err(1, "calloc");
#pragma omp parallel num_threads(2)
  foo(p);
  printf("*p = %d\n", *p);
  return 0;
}
```

(O4)

```
void foo(int * p) {
  *p += 1;
}
int main() {
  int x = 0;
#pragma omp parallel num_threads(2)
  foo(&x);
  printf("x = %d\n", x);
  return 0;
}
```

(O5)

```c
void foo(int y) {
  int * p = &y;
  *p += 1;
}
int main() {
  int x = 0;
#pragma omp parallel num_threads(2)
  foo(x);
  printf("x = %d\n", x);
  return 0;
}
```

(P1)

```c
int x = 0;
void foo() {
  x++;
}
int main() {
  pid_t pid = fork();
  if (pid == -1) err(1, "fork");
  foo();
  if (pid) {
    pid_t qid = waitpid(pid, 0, 0);
    if (qid == -1) err(1, "waitpid");
    printf("x = %d\n", x);
  }
  return 0;
}
```

(P2)

```c
void foo(int * p) {
  *p += 1;
}
int main() {
  int * p = (int *)calloc(1, sizeof(int));
  if (!p) err(1, "calloc");
  pid_t pid = fork();
  if (pid == -1) err(1, "fork");
  foo(p);
  if (pid) {
    pid_t qid = waitpid(pid, 0, 0);
    if (qid == -1) err(1, "waitpid");
    printf("*p = %d\n", *p);
  }
  return 0;
}
```

(P3)

```c
void foo(int * p) {
  *p += 1;
}
int main() {
  int * p = (int *)mmap(0, sizeof(int), PROT_READ|PROT_WRITE, MAP_SHARED|MAP_ANONYMOUS, -1, 0);
  if (p == MAP_FAILED) err(1, "mmap");
  pid_t pid = fork();
  if (pid == -1) err(1, "fork");
  foo(p);
  if (pid) {
    pid_t qid = waitpid(pid, 0, 0);
    if (qid == -1) err(1, "waitpid");
    printf("*p = %d\n", *p);
  }
  return 0;
}
```

(P4)

```c
void foo(int * p) {
  *p += 1;
}
int main() {
  int * p = (int *)mmap(0, sizeof(int), PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0);
  if (p == MAP_FAILED) err(1, "mmap");
  pid_t pid = fork();
  if (pid == -1) err(1, "fork");
  foo(p);
  if (pid) {
    pid_t qid = waitpid(pid, 0, 0);
    if (qid == -1) err(1, "waitpid");
    printf("*p = %d\n", *p);
  }
  return 0;
}
```

問題 2

グレースケール画像を格納するファイル形式とそれを読み出すプログラムについて考える. グレースケール画像は, 単純には 2 次元の整数配列で表すことができる. 配列の各要素がある画素の画素値 (グレースケールなのでひとつの整数値) になっている.

これを実現する単純なファイル形式として, Portable Graymap (PGM) という形式がある. そこでは, 幅W画素, 高さH画素, 画素値として取りうる値が 0 以上M以下であるような画像を,

```
P2
W H
M
```

$$
\begin{array}{cccc}
a_{0,0} & a_{0,1} & \cdots & a_{0,W-1} \\
a_{1,0} & a_{1,1} & \cdots & a_{1,W-1} \\
\vdots & \vdots & \ddots & \vdots \\
a_{H-1,0} & a_{H-1,1} & \cdots & a_{H-1,W-1}
\end{array}
$$

のように表す. ただし, $a_{i,j}$ は i 行 j 列の画素値を表す. ここで行は一番上の行から順に 0 行, 1 行, ..., と呼び, 列は一番左の列から順に 0 列, 1 列, ... と呼ぶ.

またファイル中で, 各数字は ASCII 文字列 (テキストエディタで普通に表示できる文字列) で表されている. PGM 形式には 2 種類あり, 画素値をバイナリ形式で表す形式もあるが, 本問で PGM 形式と言えば ASCII 文字列を使った形式のことであるとする. さらに本問では以下を仮定して良い.

- $M = 255$.
- 各画素値は必要ならば空白を入れてちょうど 3 バイト (右詰め) の文字列で表されている.
- ある画素を表す 3 バイトの文字列と, 次の画素を表す 3 バイトの文字列との間に 1 バイトの隙間が入っている.
- その隙間は, 同じ行内の要素間については空白 (`' '`) であり, ある行の最後の要素と次の行の最初の要素間については改行文字(`'\n'`)である.

例えば,

という画像を表す, 上記の仮定を満たした PGM 形式のファイル — `i.pgm` とする — の中身は以下のようである.

```
P2
22 39
255
  3   3   3   3   3   3   3   3   3   3   3   3  39 113 143 167 147 118  38   3   3   3
  3   3   3   3   3   3   3   3   3   3   4 140 249 252 252 252 252 252 252 156   5   3
  3   3   3   3   3   3   3   3   3   3  96 252 252 252 252 252 252 252 252 252  65   3
  3   3   3   3   3   3   3   3   3   3 166 252 252 252 252 252 252 252 252 252  78   3
  3   3   3   3   3   3   3   3   3   3 143 252 252 252 252 252 252 252 252 238  20   3
  3   3   3   3   3   3   3   3   3   3  26 201 252 252 252 252 252 252 214  62   3   3
  3   3   3   3   3   3   3   3   3   3   3   3  45 103 126 129 105  50   3   3   3   3
  3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3
  3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3
  3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3
  3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3
  3  25 120 120 120 120 120 120 120 120 120 120 120 120 120 120 120 120 120  41   3   3   3   3
  3 120 252 252 252 252 252 252 252 252 252 252 252 252 252 252 252 252 246  23   3   3   3
  3 117 154 169 184 227 252 252 252 252 252 252 252 252 252 252 252 252 172   3   3   3   3   3
  3   3   3   3   3   3  47 221 252 252 252 252 252 252 252 252 252 252  80   3   3   3   3   3
  3   3   3   3   3   3   3 130 252 252 252 252 252 252 252 252 231   8   3   3   3   3   3   3
  3   3   3   3   3   3   3 160 252 252 252 252 252 252 252 252 144   3   3   3   3   3   3   3
  3   3   3   3   3   3   6 227 252 252 252 252 252 252 252 252  52   3   3   3   3   3   3   3
  3   3   3   3   3   3  74 252 252 252 252 252 252 252 252 211   3   3   3   3   3   3   3   3
  3   3   3   3   3   3 166 252 252 252 252 252 252 252 252 119   3   3   3   3   3   3   3   3
  3   3   3   3   3  17 242 252 252 252 252 252 252 252 252 250  31   3   3   3   3   3   3   3
  3   3   3   3   3 101 252 252 252 252 252 252 252 252 184   3   3   3   3   3   3   3   3   3
  3   3   3   3   3 193 252 252 252 252 252 252 252 252  92   3   3   3   3   3   3   3   3   3
  3   3   3   3  35 251 252 252 252 252 252 252 252 252 240  15   3   3   3   3   3   3   3   3
  3   3   3   3 128 252 252 252 252 252 252 252 252 157   3   3   3   3   3   3   3   3   3   3
  3   3   3   4 217 252 252 252 252 252 252 252 252  65   3   3   3   3   3   3   3   3   3   3
  3   3   3  56 252 252 252 252 252 252 252 252 252 221   5   3   3   3   3   3   3   3   3   3
  3   3   3 136 252 252 252 252 252 252 252 252 131   3   3   3   3   3   3   3   3   3   3   3
  3   3   3 215 252 252 252 252 252 252 252 252 252  49   3   3   3   3   3   3   3   3   3   3
  3   3  37 252 252 252 252 252 252 252 252 221   3   3   3   3   3   3   3   3   3   3   3   3
  3   3 104 252 252 252 252 252 252 252 252 150   3   3   3   3   3   3   3   3   3   3   3   3
  3   3 166 252 252 252 252 252 252 252 252  95   3   3   3   3   3   3   3   3   3   3   3   3
  3   3 205 252 252 252 252 252 252 252 252  68   3   3   3   3   3   3   3   3   3   3   3   3
  3   3 238 252 252 252 252 252 252 252 252  77   3   3   3   3   3   3   3   3   3   3   3   3
  3   3 237 252 252 252 252 252 252 252 252 169   4   3   3   3   3   5  55 129 131   5   3   3
  3   3 164 252 252 252 252 252 252 252 252 252 202 135 124 140 178 230 252 252 252 129   3   3
  3   3  29 223 252 252 252 252 252 252 252 252 252 252 252 252 252 252 252 246 177  73   4   3
  3   3   3  30 172 251 252 252 252 252 252 252 252 252 252 238 181 101  21   3   3   3   3   3
  3   3   3   3   3  33 102 143 162 164 145 128 102  46   5   3   3   3   3   3   3   3   3
```

以下の問いに答えよ. なお参考までに以下の 2 つの関数を含んだファイル `pgm.c` と上記の `i.pgm` を配布している. `pgm.c` はあくまでファイル形式の説明の補足として示しているもので, 解答にこれをそのまま使えという意味ではない. 使ってもよいが仮に間違いがあったらそれを適宜修正して使うのも解答者の責任である.

- `char * read_pgm(char * a_pgm, long * Wp, long * Hp, long * Mp)`

  引数 `a_pgm` で与えられた名前の PGM ファイルを読み込む. ファイルが読み込めなかったり画素数が足りないなどエラーが生じた場合はプログラムを終了させる. 成功した場合, 幅 (Wとする), 高さ (H とする), 画素値の最大値 (M とする. 本問では常に 255) をそれぞれ `*Wp`, `*Hp`, `*Mp` に格納する. 返り値として画素値の配列 (の先頭へのポインタ) を返す. 具体的には, 返り値をaとして, i 行 j 列の画素値が $a[iW + j]$ に格納された配列を返す.

- `void write_pgm(long W, long H, long M, char * a, char * a_pgm)`

  幅が `W` 画素, 高さが `H` 画素, 画素値の最大値が `M`, 画素値が配列 `a` で与えられる PGM ファイルを引数 `a_pgm` で指定された名前で作る. i 行 j 列の画素値が `a[`i `W +` j`]` で与えられる. ファイルが作れないなどエラーが生じた場合はプログラムを終了させる.

(1) 幅W画素, 高さH画素の画像を PGM 形式で表すのに何バイト必要か? ただし PGM 形式の画素値の部分だけを数えればよく, ヘッダ部分:

```
P2
W  H
M
```

は含めなくて良い.

(2) 以下のように PGM 形式のファイル名 f と, 整数 i, j をコマンドラインで与えられると,

`./pixel_val` f i j

f の i 行 j 列の画素値を表示するプログラム `pixel_val` を C 言語で書け. 例えば上記の `i.pgm` に対して期待される出力は以下の通り (`$` は, シェルのコマンドプロンプトでありコマンドの一部ではない).

```
$ ./pixel_val i.pgm 0 0
3
$ ./pixel_val i.pgm 11 2
120
```

ただしプログラムは**効率的**なものでなくてはならず, それは実行時間がファイルサイズにほとんどよらないことを意味する. 解答欄にあるコマンドでコンパイルし, テストまで行うこと. コンパイルのためのコマンドは必要ならば変更しても良い.

(3) PGM 形式のファイル名fと, 以下のように行と列の組が 1 行にひとつずつ並んだファイル (**添字ファイル**と呼ぶことにする) I

$$\begin{matrix} i_0 & j_0 \\ i_1 & j_1 \\ \vdots & \vdots \\ i_{N-1} & j_{N-1} \end{matrix}$$

が与えられると, 指定されたすべての画素値の合計値, すなわち $a_{i_0,j_0} + a_{i_1,j_1} + ... + a_{i_{N-1},j_{N-1}}$ を計算して表示するコマンドを, `pixel_val` を指定された画素に対して順に呼び出すだけのシェルスクリプト `pixel_vals.sh` として書け. 例えば上記の `i.pgm` と, 以下の内容を持つ添字ファイル `idx.txt`

```
0 0
11 2
```

に対して,

```
$ ./pixel_vals.sh i.pgm idx.txt
```

のように起動ししたときに期待される出力は, `123` である ($3 + 120 = 123$ だから).

シェルスクリプトをファイルに保存し, 解答用紙にあるコマンドでテストまで行うこと. `awk` コマンドを使うと良いかもしれない (使わなくても構わない). ヒントとして以下は, `idx.txt` を第一引数として与えるとその内容を順に表示するだけのシェルスクリプトである.

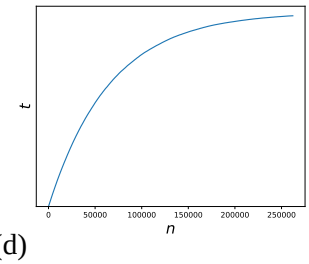
```
#!/bin/bash

cat $1 | while read i j ; do
    echo "i=$i j=$j"
done
```

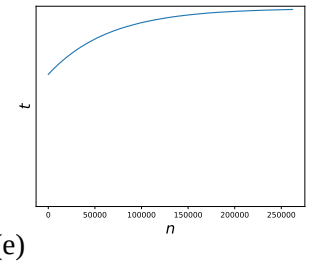
(4) $W = H = 2^{14} (= 16384)$ である大きな PGM ファイルがある. このファイルがキャッシュ上にない状態で, 行と列が $N = 2^{18}$ 個入った添字ファイルを上記のシェルコマンドに与える. ただし各行, 列の番号は 0 以上 16384 未満の数がランダム (一様) に選ばれているとする. このプログラムの進行を, 横軸を読み出した画素数 (n), 縦軸を n 画素読み出すまでに

かかった時間 (t) としてグラフに書くとどのようになるか? 以下から適切なものを 1 つ選び, そのようになると考える理由も述べよ. なお, 複数のグラフの縦軸は同じスケールで描かれているとは限らない.

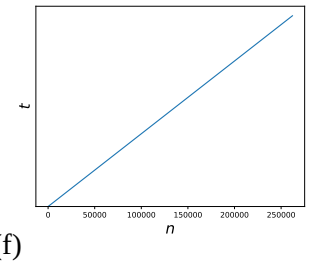
(a)



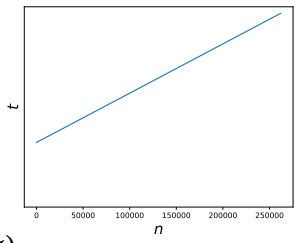
(b)



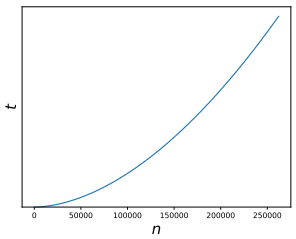
(c)



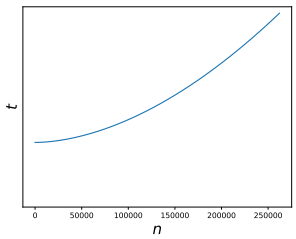
(d)



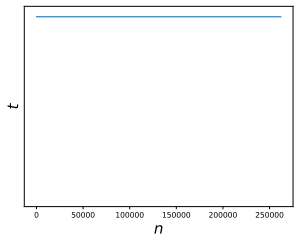
(e)



(f)



(g)



(5) 上記と同等の動作をするプログラムをさらに高速化したい. 考えられる方針を 1 つ述べ, なぜそれで高速化するのかを述べよ.

(6) 上記で述べた方法のうちの一つを実際にプログラムにせよ. プログラムは一つの C 言語のプログラムとして, `pixel_vals.c` というファイルに書け. 解答用紙にあるコマンドラインでコンパイルとテストを行え. コンパイルのためのコマンドは必要ならば変更しても良い.

問題 3

次のような, 同期のためのデータ構造 gate_t と関連する関数を考える. gate_t は mutex (排他制御) と似ているが, l を初期化時に定めた定数として, クリティカルセクションに同時に l スレッドまで同時に入ることを許す. 例えて言うならば, l 台まで自動車が止められる駐車場の入口の門番を行うデータ構造である.

具体的には以下の 3 つの関数を持つ.

- `void gate_init(gate_t * g, uint32_t l);`
- `void gate_enter(gate_t * g);`
- `void gate_leave(gate_t * g);`
- 注: `uint32_t` は 32 bit の符号なし整数の型である. 符号なしであることに深い意味はないが, 後に `futex` 関数に渡す都合上そのようにしている.

`gate_init(g, l)` を一度だけ呼び出し, あとはそのgに対して各スレッドが, `gate_enter(g)` と `gate_leave(g)` を交互に呼び出す.

このとき gate_t は, `gate_enter(g)` の呼び出しを終えて `gate_leave(g)` をまだ呼び出していないスレッドの数が l 以下であることを保証する.

より具体的には以下のようなコードを動かした際, ???の部分を実行中のスレッドが常に l 以下であることを保証する.

```
gate_t g[1];
gate_init(g, l);
#pragma omp parallel
{
  while (...) {
      ...
    gate_enter(g);
      ???
    gate_leave(g);
      ...
  }
}
```

以下の問いに答えよ.

(1) gate_t と上記の 3 つの関数を, mutex や, その他必要な Pthread の同期のためのプリミティブを用いて実現せよ.

(2) それらを compare_and_swap 以外の同期プリミティブを一切用いずに実現せよ. ただしビジーウェイトをしてもよい. なお解答欄に compare_and_swap 命令の定義が書いてあるので使って良い (授業で述べたときと用いている関数名が変わっているが, GCC の仕様変更によるものである).

(3) それらを, compare_and_swap と futex を用いて, mutex を用いずに実現せよ. ビジーウェイトしてはならない. ここで「ビジーウェイトしない」とは直感的には, あるスレッドAが, 別のスレッド B が進行するのを待つだけのために長時間 CPU を使わない (待つならばブロックする) ということだがもう少し厳密な定義を述べると, 「任意のタイミングで任意

個のスレッドを停止させたとしても, その状態で残りのスレッドは有限時間で進行するか, さもなければブロックする」ということである.

さて, 仕様を少し変更して, `gate_enter` が `uint32_t` の値を返し, `gate_leave` は `uint32_t` の値を受け取ることにする.

- `void gate_init(gate_t * g, uint32_t l);`
- `uint32_t gate_enter(gate_t * g);`
- `void gate_leave(gate_t * g, uint32_t x);`

`gate_enter` の返り値は 0 以上 l 未満の整数で, 駐車場の比喩を続けるならば, どの駐車スロット空いているかを返す. もう少し厳密に言うと以下の通り:

- あるスレッドが `gate_enter(`g`)` を呼び出して返り値 x が返されたあと, まだそのスレッドが `gate_leave(`g`)` を呼んでいない状態を, 「そのスレッドがg から x を受け取っている」ということにする.
- g から x を受け取っているスレッドが `gate_leave(`g`, ..)` を呼び出す際は, その x を第 2 引数として渡さなくてはならないものとする.
- その仮定のもとで, 複数のスレッドが同じ `gate_t` から同じ値を受け取っていることはない.

(4) このように仕様を変更した `gate_t` と 3 つの関数を, mutex, その他必要な Pthread のプリミティブを用いて実現せよ.

(5) それらを, `compare_and_swap` と `futex` を用いて, mutex を用いずに実現せよ. ビジーウェイトしてはならない.

問題は以上である.