OS1 H27 年度 後期末試験

(2016.02.09 重村 哲至)

IE4 ____ 番 氏名 ____模範解答____

1. 次の文章の空欄に適切な言葉を、語群から記号で答えなさい。(2 点× 20 問= 40 点)

ユーザプログラムに使いやすいプロセス間通信を提供するためにメッセージ転送機構が提供される。これには、プロセス番号などを用いて通信相手を指定する **(1)** 指定方式と、**(2)** 等の名前を用いて指定する **(3)** 指定方式のものがある。**(3)** 指定方式の方が **(4) と** の通信を自然に実現できる。

入出力同期などで複数のイベントを同時に待ち合わせ、どれかのイベントが発生すれば先に進めるような場合を **(5)** 条件と言う。ダイニングフィロソファ問題のように複数の資源を同時に使用するために、全ての資源が揃うまで先に進めないような場合を **(6)** 条件という。ダイニングフィロソファ問題等でデッドロックが発生するのは、資源の一部を確保したまま **(7)** 状態になることが原因である。資源の確保 **(8)** に制約を設けることで解決する場合もあるが一般的な解ではない。必要な複数資源を同時に確保する P 命令の拡張版 **(9) か** 命令を導入すると良い。

シーケンサとイベントカウントはプロセス統合機構の一種である。シーケンサ (S) には新しい値を返す **(10)** 操作が許される。イベントカウント (E) には、E の値が目的の値になるまで待つ **(11)** 操作、E の値を進めて E を待っていたプロセスを起こす **(12)** 操作、E の値を読み取る **(13)** 操作が許される。**(10)** 操作は **(14)** のシステムでは **(15)** にすることで簡単に実現できる。一方、**(16)** のシステムでも専用の機械語命令を用いれば効率的に実現することができる。

(17) はリソース管理の為の機能と制約をもつ抽象データ型である。**(18)** の働きにより複数のプロセスが同時に **(17)** に入ることはできない。プロセスが **(19)** の待ち行列に入る（待ち状態になる）と、**(18)** が解除され次のプロセスが **(17)** に入ることが可能になる。**(19)** の待ち行列を起こすのは **(17)** に入った別のプロセスである。待ち行列のプロセスは起こされると **(20)** 実行され、その後、起こしたプロセスの実行に戻る。その間、**(18)** は一度も解除されない。

語群：(あ) AND、(い) advance、(う) await、(え) NOT、(お) OR、(か-1) P_and、(か-2) P_or、(き) read、(く) ticket、(け) write、(こ) XOR、(さ) ガード、(し) セマフォ、(す) マルチプロセッサ、(せ) モニタ、(そ) ユニプロセッサ、(た) リンク、(ち) 間接、(つ) 条件変数、(て) 順序、(と-1) 1 対 1 、(と-2) 多対多、(な) 直ちに、(に) 直接、(ぬ) 待ち、(ね) 割込許可、(の) 割込禁止

(1)	(に)	(2)	(た)	(3)	(ち)	(4)	(と-2)	(5)	(お)
(6)	(あ)	(7)	(ぬ)	(8)	(て)	(9)	(か-1)	(10)	(く)
(11)	(う)	(12)	(い)	(13)	(き)	(14)	(そ)	(15)	(の)
(16)	(す)	(17)	(せ)	(18)	(さ)	(19)	(つ)	(20)	(な)

2. 次の C 言語風のプログラムは、複数プロデューサ／複数コンシューマ問題のシーケンサとイベントカウントを用いた解を示しています。空欄に適切な記述を答えなさい。(20 点)

```
#define N 5                  // バッファの大きさ

SEQUENCER  T = 0;            // 初期値 0 のシーケンサ (T)
SEQUENCER  U = 0;            // 初期値 0 のシーケンサ (U)
EVENTCOUNT IN = 0;           // 初期値 0 のイベントカウント (IN)
EVENTCOUNT OUT = 0;          // 初期値 0 のイベントカウント (OUT)
MESSAGE    BUF[N];           // 大きさ N のバッファ(資源)

producer() {                 // 生産者プロセスが実行する関数
  int t;
  while (true) {
    t=ticket(T);
    await(IN, /*(A)*/);      // 空欄 (A) を埋める (他の生産者と同期をとる)
    await(OUT,/*(B)*/);      // 空欄 (B) を埋める (バッファの空きを待つ)
    BUF[t%N]=メッセージ;
    advance(IN);
  }
}

consumer() {                 // 消費者プロセスが実行する関数
  int u;
  while (true) {
    u=ticket(U);
    await(OUT,/*(C)*/);      // 空欄 (C) を埋める (他の消費者と同期をとる)
    await(IN, /*(D)*/);      // 空欄 (D) を埋める (メッセージの到着を待つ)
    メッセージ=BUF[u%N];
    advance(/*(E)*/);        // 空欄 (E) を埋める
  }
}
```

(A)	t	(B)	t-(N-1)
(C)	u	(D)	u+1
(E)	OUT		

3. 次の C 言語風のプログラムは、リーダ・ライタ問題のシーケンサとイベントカウントを用いた解を示しています。空欄に適切な記述を答えなさい。(20 点)

```
SEQUENCER  W = 0;        // 初期値 0 のシーケンサ (W)
EVENTCOUNT X = 0;        // 初期値 0 のイベントカウント (X)
writer() {               // 複数のライタプロセスが実行する関数
  int w;
  while (true) {
    w=ticket(W);
    await(X, w);
    データ書き込み ();   // 自分の順番が来たらデータを書き込む
    advance(X);
  }
}

SEQUENCER  M = 0;        // 初期値 0 のシーケンサ (M)
EVENTCOUNT Y = 0;        // 初期値 0 のイベントカウント (Y)
int        R = 0;        // リーダ間の共有変数
reader() {               // 複数のリーダプロセスが実行する関数
  int m, r;
  while (true) {
    m=ticket(M);
    await(/*(A)*/, m);
    if (R==0) {          // 入口では最初のプロセスがリーダを代表
      r=ticket(W);
      await(/*(B)*/, r);
    }
    R++;
    advance(/*(C)*/);
    データ読み出し ();
    m=ticket(M);
    await(Y, m);
    R--;
    if (R==0)            // 出口では最後のプロセスがリーダを代表
      advance(/*(D)*/);
    advancd(Y);
}
```

(A)	Y	(B)	X
(C)	Y	(D)	X

4. 次の Java プログラムは、プロデューサ／コンシューマ問題の解を示しています。下の問に答えなさい。(20 点)

```
class ThreadBuffer {                        // 同期機能付きのバッファクラス
  final static int n = 5;                   // バッファの大きさ
  int   p = 0;                              // 次にデータを入れる場所
  int   q = 0;                              // 最後のデータが入っている場所
  int[] buf = new int[n];                   // 大きさ N のバッファ(資源)
  private int next(int ptr) {               // バッファで次の位置を求めるメソッド
    if (++ptr >= n) return 0;
    return ptr;
  }
  public synchronized void in(int x) { // 生産者プロセスが呼出す
    while (next(p)==q)                      // キューがいっぱいの場合は待つ
      try { wait(); } catch(InterruptedException e) {}
    buf[p] = x;
    p = next(p);
    notify();
  }
  public synchronized int out() {       // 消費者プロセスが呼出す
    while (p==q)                            // バッファが空の間は待つ
      try { wait(); } catch(InterruptedException e) {}
    int r = buf[q];
    q = next(q);
    notify();
    return r;
  }
}
```

(a) なぜ wait() は while ループの中で実行する必要があるか？理由を説明しなさい。

wait() は notify() 以外でも終了することがある。notify() 以外で終了した場合は再度 wait() する必要がある。

(b) このプログラムは複数プロデューサ／複数コンシューマ問題の解になるか？
なるかならないか答えた上で理由も説明しなさい。

複数プロデューサ／複数コンシューマ問題の解になる。in() ／ out() は他のプロデューサ／コンシューマとの排他も行うので、複数プロデューサ／複数コンシューマ問題の解にもなっている。