

並行プロセス

佐藤一郎

2001年7月10日

Ichiro Satoh

並行プロセス

並行(並列)に動作するプロセスは相互に作用しあいながら動作可能

■ 情報交換:

あるプロセスから別のプロセスに計算結果を渡す



■ 同期:

あるプロセスが所定の条件を満足するまで別の一方のプロセスを休止



Ichiro Satoh

並行プロセス

並行(並列)に動作するプロセスは相互に作用しあいながら動作可能
競合(competition)

複数プロセスが共用する資源を同時に使用しようとする状態
→ 相互排除(共用資源は高々一つのプロセスしか使用させない)

協調(cooperation)

複数プロセスが協力して共通の目的のために処理を行う形態



Ichiro Satoh

並行処理における共有変数

プログラム中の一つ以上の部分が同時に処理されると変数内容が意図した内容と違うことがある

```
int x = 0;
program {
  thread A {
    ...
    x = x + 3;
    ...
  }
  thread B {
    ...
    x = x + 4;
    ...
  }
}
```

x = 7のとき
x=x+3とx=x+4が重ならず実行

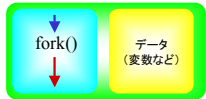
x=3またはx=4のとき
x=x+3とx=x+4が重なって実行

Ichiro Satoh

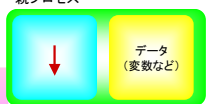
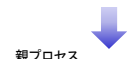
UNIXのプロセス生成

forkシステムコールによりプロセスを生成
メモリ空間ごとコピー

プロセス



```
#include <sys/types.h>
#include <unistd.h>
pid_t fork(void);
int main(void) {
    int pid;
    pid = fork();
    if (pid < 0) {
        printf("error");
    }
    if (pid == 0) {
        printf("I am a child");
    }
    else {
        printf("I am a parent");
    }
    return 0;
}
```



Ichiro Satoh

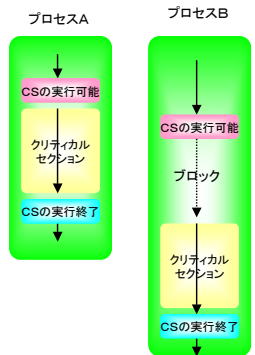
クリティカル領域

きわどい領域 (Critical Region)
複数のプロセス (スレッド) 同時に
実行してはいけないプログラム部分

排他的なロックにより実行を制御

ロックの実現方法:

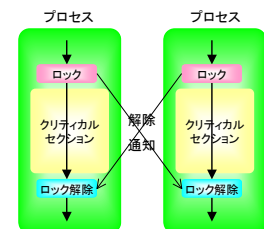
- ビジーウェイティング
- セマフォ
- モニター



Ichiro Satoh

ロックによる相互排除

あるプロセスが共有資源にアクセスしていることを表明し、他のプロセス
のアクセスを制限するために機構



制限中のプロセスは待機

クリティカルセクション:

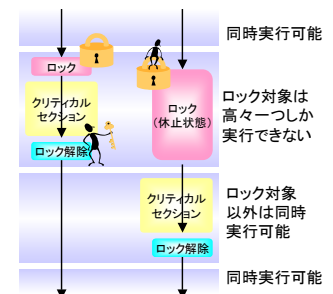
複数プロセスの同時実行により障害が発生する処理

Ichiro Satoh

ロックによる相互排除

競合を回避するために複数プロセスの特定処理の同時実行を禁止

- あるプロセスが共有資源をロックしているとき他のプロセスは共有資源をロックすることができず、待機状態(ブロック)になる
- ロックを獲得しているプロセスがロックを解除すると、ロックを待ちのプロセスのうち一つがロック可能になり、処理が再開可能



Ichiro Satoh

同期の実現

排他的なロックの実現方法

- ビジーウェイトによるロック
プロセス間の共用変数の状態によりロック獲得プロセスを決める
- セマフォによるロック
ロック待ち合わせ命令とロック解除の通知命令を用意
- モニタによるロック
共用資源をアクセスするための手続きを高々一つしか呼び出せないように制限

Ichiro Satoh

同期の実現(ビジーウェイト)

ビジーウェイトによるロック

- ある条件が成立しているかを常に監視する
- 条件成立後にクリティカルセクションを実行
- クリティカルセクション終了直前に条件を変更して他のプロセスが実行可能にする。

プロセスA

```
While (true) {  
    while (turn != A); // loop  
    enter_critical_region();  
    // 排他的な実行箇所  
    turn = 'B';  
    exit_critical_region();  
}
```

プロセスB

```
While (true) {  
    while (turn != B); // loop  
    enter_critical_region();  
    // 排他的な実行箇所  
    turn = 'A';  
    exit_critical_region();  
}
```

Ichiro Satoh

同期の実現(セマフォ)

セマフォ(semaphore)

- wait(semaphore): semaphoreが真(または正数)ならば、その値を1減らしてwait()以降のプログラム実行を許す。semaphoreが偽(または0)ならばそのプロセスをブロックする。
- signal(semaphore): semaphoreを真(または1増加)させて、ブロック中のプロセスがwait()を実行することを許す。

プロセスA

```
wait(semaphore);  
// 排他的な実行箇所  
signal(semaphore);
```

プロセスB

```
wait(semaphore);  
// 排他的な実行箇所  
signal(semaphore);
```

Ichiro Satoh

同期の実現(モニタ)

モニタ(monitor)

- 内部では高々1つの手続きだけを実行
- 実行条件が成立しないときはブロックされる
- 実行中のプロセスは条件の変更を明示的に通知可能

```
monitor Program {  
    int top = 0;  
    int stack[MAX];  
    procedure void push(int x) {  
        if (top == MAX) { wait(full); }  
        top = top + 1;  
        stack[top] = x;  
        if (top == 0) { signal(empty); }  
    }  
    procedure int pop() {  
        if (top == 0) { wait(empty); }  
        int item = stack[top];  
        top = top - 1;  
        return top;  
        if (top == MAX-1) { signal(full); }  
    }  
}
```

Ichiro Satoh

プロセス間通信

プロセス間で協調を行うには同期をとるだけでなく、プロセス間で情報の受け渡しが必要となる。

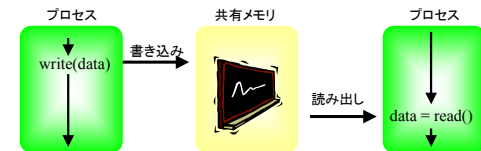
代表的なプロセス間通信方法

- 共有メモリ(shared memory)
- メッセージ通信(message passing)
 - 非同期メッセージ(asynchronous communication)
 - 同期通信(synchronous communication)

Ichiro Satoh

共有メモリ

複数プロセス間で共有するメモリをもち、そのメモリにデータを読み書きして情報を交換



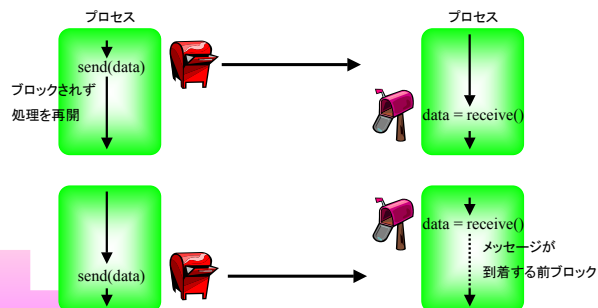
読み手側プロセス及び書き手側プロセスとともにブロック
されることはない

Ichiro Satoh

非同期通信

あるプロセスから他のプロセスにメッセージを送信

- 送信側プロセスは送信命令実行後にブロックされない
- 受信側プロセスは受信命令実行後にメッセージが届くまでブロック

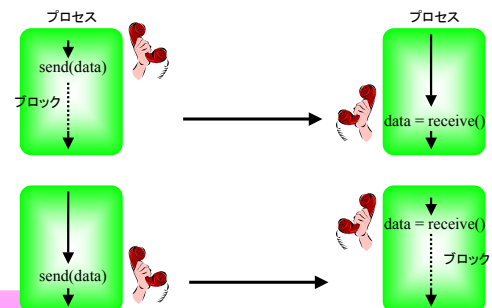


Ichiro Satoh

同期通信

あるプロセスから他のプロセスにメッセージを送信

- 送信側プロセスは受信側プロセスが通信可能になるまでブロック
- 受信側プロセスは送信側プロセスが送信可能になるまでブロック



Ichiro Satoh