モニタを用いたアルゴリズム

1. モニタを用いた相互排除アルゴリズム

モニタ

```
monitor {
    private condition resource;              /* 条件変数 */
    private int s := 1;                      /* 空き資源の数 */

    public void csBegin () {                 /* 臨界領域開始処理 */
        if (s=0) resource.wait;      /* 資源を要求 空き資源が無ければ空くまで待つ */
        --s;
    }

    public void csEnd() {                    /* 臨界領域終了処理 */
        ++s;
        resource.signal;                     /* 資源解放 */
    }
}
```

プロセス *i* の相互排除処理

```
monitor.csBegin();                           /* 資源を要求する */
CSi();                                       /* プロセス i の臨界領域 */
monitor.csEnd ();                            /* 資源を解放する */
NCSi();                                      /* プロセス i の非臨界領域 */
```

2. モニタを用いたパイプ処理

モニタ

```
monitor {
    private condition empty, full ;          /* 条件変数 */
    private int i ;                          /* バッファへの次の書き込み位置 */
    private int j ;                          /* バッファからの次の読み出し位置 */
    private int m ;                          /* バッファ中のメッセージ数 */
    private Message Buffer[N];               /* バッファ */

    public void put (Message msg) {          /* バッファへの書き込み */
        if (m ≧ N) full.wait;                /* バッファがいっぱいなら待つ */
        Buffer[i] := msg; i := (i +1) mod N; m++;      /* 書き込み */
        empty.signal;
    }

    public Message get() {                   /* バッファからの読み出し */
        if (m = 0) empty.wait;               /* バッファが空なら待つ */
        msg := Buffer[j];   j := (j +1) mod N; m--;    /* 読み出し */
        full.signal;
        return msg;
    }
}
```

送信側プロセスの送信処理

```
while (true) {
    send_msg の生成;
    monitor.put (send_msg);
}
```

受信側プロセスの受信処理

```
while (true) {
    recv_msg := monitor.get();
    recv_msg の処理;
}
```

3. モニタを用いたリーダライタ問題アルゴリズム

モニタ

```
monitor {
    condition OkWrite, OkRead;   /* 条件変数 */
    private int w := 0;          /* 書き込みプロセス数 */
    private int r := 0;          /* 読み出しプロセス数 */

    public void writeBegin() {   /* 書き込み開始 */
        if ((r > 0) or (w>0))
            OkWrite.wait;        /* 他のプロセスが読み/書き中なら待つ */
        ++w;
    }

    public void writeEnd() {     /* 書き込み終了 */
        --w;
        if (OkRead.queue)
            OkRead.signal;       /* 読み出し待ちプロセスがいれば読み出し可能に */
        else OkWrite.signal;     /* いなければ書き込み可能に */
    }

    public void readBegin() {    /* 読み出し開始 */
        if ((w > 0) or OkWrite.queue) /* 書き込み中か書き待ちプロセスがいれば
            OkRead.wait;             /* 読み出し待ちに */
        ++r;
        while (OkRead.queue)     /* 読み出し待ちプロセスを全て読み出し可能に */
            OkRead.signal;
    }

    public void readEnd () {     /* 読み出し終了 */
        --r;
        if (r =0) OkWrite.signal;
    }
```

ライタプロセスの書き込み処理

```
while (true){
    write_data の生成;
    monitor.writeBegin();
    write(write_data);
    monitor.writeEnd();
}
```

リーダプロセスの読み込み処理

```
while (true) {
    monitor.readBegin();
    read_data := read();
    monitor.readEnd();
    read_data の処理;
}
```