リアルタイムシステム

第一回レポート課題

課題内容

「哲学者の食事問題」に関して、テキスト P.27 の例より効率のよいアルゴリズムはどういうものか説明せよ（出来れば擬似コードを記載すること）。

問題点

テキスト P.27 に記載されている擬似コードを以下に示す。

```
for ( ; ; )
{       /* 思索する */
        wai_sem ( room ) [TF2] ;
        wai_sem ( 右のフォーク ) ;
        wai_sem ( 左のフォーク ) ;
        /* 食事する */
        sig_sem ( 右のフォーク ) ;
        sig_sem ( 左のフォーク ) ;
        sig_sem ( room ) ;
}
```

■ 計数セマフォ

この擬似コードの例では、計数セマフォ room（初期値＝最大値＝5-1＝4）により、協調制御が行われており、5 人の哲学者が同時に食事に取り掛かろうとしたとき、最後に食事に取り掛かろうとした哲学者を待たせる処理をしている。

この処理により、同時に 4 人の哲学者までしか食事に取り掛かれず、全員が同じ方向のフォークを持ち、誰も食事に取り掛かれない状況（デッドロック）を防止している。

■ バイナリセマフォ

各フォークに対応するセマフォは初期値が 1 のバイナリセマフォである。このセマフォは、複数の哲学者がひとつのフォークを共有しないように排他制御を行っている。

■ 効率の問題

このテキストの例では、最低でも 1 人の哲学者が食事をすることができ、かつ、デッドロックにより全員が餓死することもない。

しかし、5 つの席があれば、常に 2 人の哲学者が食事できる状態がリソースを最も効率よく使用している状態といえる。この擬似コードがその状態を満たしているとはいいがたい。

■ 理想的な処理状態

ここで、ひとつの例を示す。哲学者 5 人（P1~P5）、フォーク 5 本（F1~F5）であるとき（図 1）、哲学者が P5→P4→P3→P2→P1 という順に食事に取り掛かる場合を考える。

理想的な処理状態の場合、まず P5 が F1,F5 を使用し食事を開始する。P4 は F5 が使えないため食事ができないが、P3 は F4,F3 を使用し食事ができる。また、P2 は F3 が使えないため食事が

できない。さらに、P1 はデッドロック防止のため、食事に取り掛かることさえできない。

次に、P4,P2 は、P5,P3 が食事を終えると同時に食事に取り掛かり、P1 は P2 が食事を終えると同時に食事に取り掛かることができる。これが理想的な流れである。

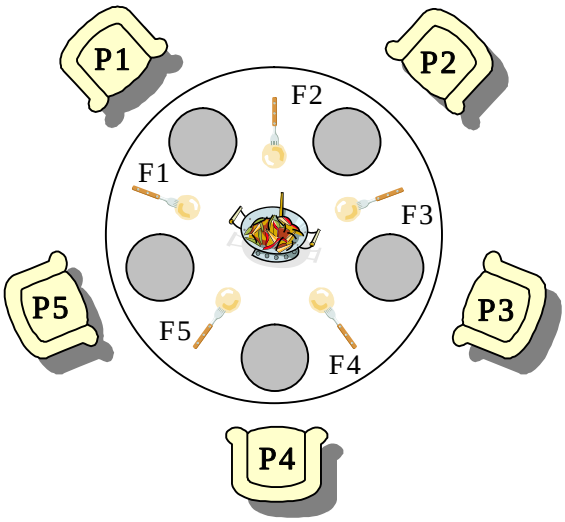

図 1　哲学者の食事問題

■ テキストの例の処理状態

一方、テキストの例の処理状態の場合、まず、P5 が F1,F5 を使用し食事を開始する。P4 は F4 をとり F5 が空くのを待つ。P3 は F3 をとり、F4 が空くのを待つ…。というようになり、P5 が食事を終えたら P4 が食事を、P4 が食事を終えたら P3 が食事を…、という具合に、理想的な状態と比べ次々と全員が食事を終える時間が遅れていくことになる。

■ 処理時間の比較

哲学者が食事にかける時間を 10 秒、前の哲学者が食事をしようとしてから、次の哲学者が食事をしようとするまで、それぞれ 1 秒かかるとした場合の結果を見てみると(図 2)、最終的に 16 秒もの遅れが生じることになる。

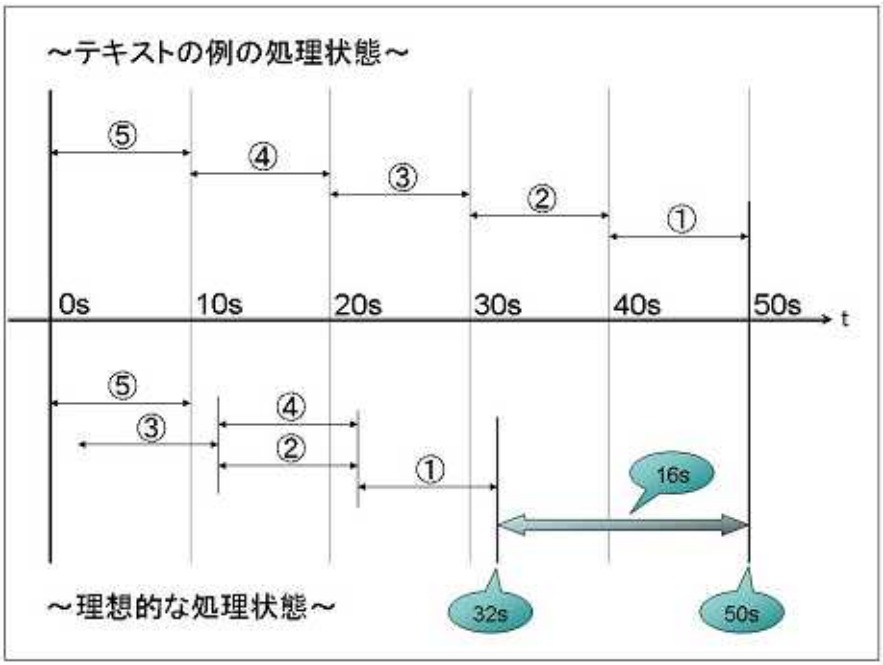

図 2　処理状態の比較

解決策

上記に示した問題を解決するための指標を以下に示していく。

■ 無駄なフォーク確保

上記の例で発生する問題の核心は、P4 が「左のフォークが使用できないのに、右のフォークを確保してしまう」ことにあると考えられる。食事を開始できないにも関わらず、別のフォークを確保してしまうことで、フォーク（資源）に無駄が生じ、結果として次の哲学者（タスク）P3 が食事できないという事態を招いてしまっている。

■ フォーク確保のタイミング

この問題を解決するためには、フォーク確保のタイミングを、

①. 左右のフォークがどちらも使用できる場合

②. 自分より後にきた哲学者が必要なフォークを使用している場合

に限り、①に場合は両方のフォークを、②に場合は使用できるフォークをそれぞれ確保すればよい。（②の場合は、例えば P5→P4→P3 と食事をする場合、P4 は自分より後に来た哲学者 P3 がまだ食事をしている時、P5 の食事が終わり次第、左のフォーク（F5）を事前に確保してよいということである。）

■ 具体策

フォーク確保のタイミングを具体的に実装するためには、左右のフォークが使用可能であるかどうかチェックし、その状況において処理を変更することが必要になってくる。例えば、左のフォークは使用できるが右のフォークは使用できない場合、必ず右のフォークを確保した後で左のフォークを確保するという処理をすべきである。こうすれば、空いている左のフォークを先に確保してしまうことがなくなるだけでなく、右のフォークが空き次第それを確保することで、ロックアウト[1]の問題も払拭できる。

左右のフォークの使用状況は 4 つのパターンが考えられ、以下にそれぞれの場合の処理を示す。

表 1　フォークの使用状況と処理

右のフォーク	左のフォーク	処理
○（使用可能）	○（使用可能）	どちらのフォークからとってもよい
○（使用可能）	×（使用不可）	左のフォークから取る
×（使用不可）	○（使用可能）	右のフォークから取る
×（使用不可）	×（使用不可）	どちらのフォークからとってもよい

[1]この例では、ある哲学者の両隣の哲学者が意地悪をして、2 人が交代で食事を取ったら、真ん中の哲学者がいつまでたっても左右のフォークが揃わず、飢えてしまうといった状況。

■ 擬似コード

上記の具体策を実装する際の擬似コードを以下に示す。

```
/* R (Right) 自分から見て右のフォークのバイナリセマフォのセマフォ変数 */
/* L (Left) 自分から見て左のフォークのバイナリセマフォのセマフォ変数 */
for ( ; ; )
{       /* 思索する */
wai_sem ( room ) ; /* 食事に取り掛かる */
if ( R == 0 && L == 1 ) { /* 右のフォーク:使用不可・左のフォーク:使用可能 */
        wai_sem ( R ) ; /* 先に右のフォークが空くまで待つ */
        wai_sem ( L ) ; /* 右を確保してから左のフォークを待つ */
} else if ( R == 1 && L == 0 ) { /* 右のフォーク:使用可能・左のフォーク:使用不可 */
        wai_sem ( L ) ; /* 先に左のフォークが空くまで待つ */
        wai_sem ( R ) ; /* 左を確保してから右のフォークを待つ */
} else { /* どちらも使用可能か、どちらも使用不可 */
        wai_sem ( R ) ; /* 左右のどちらが先でもよい */
        wai_sem ( L ) ; /* 左右のどちらが先でもよい */
}
/* フォークが確保できたので、食事を行う */
sig_sem ( R ) ; /* 右のフォークを開放 */
sig_sem ( L ) ; /* 左のフォークを開放 */
sig_sem ( room ) ; /* 食事を終了し、思索に戻る */
}
```

■ 動作

この擬似コードを先ほど示した例（哲学者 5 人・フォーク 5 本）を元に動作確認してみる。まず、P5 が食事に取り掛かりる。左右のフォークが使用可能であるので、両方のフォークを確保し食事を始める。P4 は、左は使用可能だが右は使用不可なので、先に左のフォークが空くまで待ち右のフォークは確保しない。P3 は P5 と同様にフォークを確保し食事を始める。P2 は P4 と同様に先に左のフォークを待つ。P1 は room が 0 なので食事に取り掛かれない。

この動作を理想的な処理状態と比較すると、なんら差異のない動作結果が得られていることがわかり、常に 2 人の哲学者が食事できる状態を確保しているといえる。

考察・参考文献

■ 考察

左右のフォークがどちらも使用可能になってから、同時に二つのフォークを取ればいいという考えにはすぐに至ったが、それを擬似コードで示す際に非常に苦労した。

最初、セマフォ変数 R と L が両方 1 の場合には、テキストのままでよいことはわかったが、そうでない場合にどのようにタスクに待ってもらうのかがわからなかった。結局、右・左のどちらが使用可能でどちらが使用不可なのかを判別することにより、先にどちらのフォークを確保すればよいのかを考えるようにし、上記のような擬似コードを考案するに至った。

■ 参考文献

以下の Web サイトを参考にした。

≪食事する哲学者≫

「http://www.tokumaru.org/techterm/philosopher.html」

≪フリー百科事典-Wikipedia≫

「http://ja.wikipedia.org/wiki/」