H24. 秋. ソフトウェア（排他制御）

問1　プロセスの排他制御（ソフトウェア）　（H24秋-FE 午後問1）

【解答】

［設問1］　イ
［設問2］　a－ア，b－オ
［設問3］　ウ

【解説】

プロセスの排他制御に関する問題である。これまでは，データベースシステムを題材にして排他制御を取り上げることが多かったが，午前試験を含めて，出題頻度の高い重要項目の一つである。排他制御の必要性（二重更新の回避）や排他制御を行った場合に生じる不具合（デッドロック）などについては，理解して試験に臨む必要がある。内容としては，比較的解きやすかったのではないかと推測している。

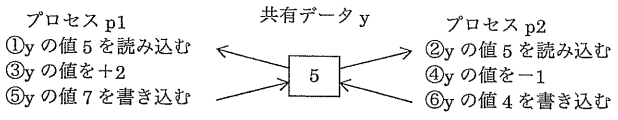
設問2では，同期変数sを用いた排他制御の内容が問われているが，セマフォ（変数）をイメージできればよい。しかし，セマフォはどちらかというと上級試験の範疇であるため，知らなかった受験者も多いかもしれない。受験者が学習済みであろうロック法でいうと，変数の値により，ロックとアンロック（ロックの解除）を制御することである。

例えば，変数sの値として，0及び1を割り当て，0のときはアンロック状態，1のときはロック状態というように，他のプロセス（プログラム）からのアクセスを制御することである。セマフォでは，ロック操作をP操作，アンロック操作をV操作として実現しており，セマフォ変数sは，アンロック状態（解放状態）が1で，ロック状態（確保状態）を0とするが，ここでは，確保／解放の区別が付けばよいのであるから，どちらでもよく，問題文にも確保状態と解放状態としか言及されていない。

設問内容は難しくないので，落ち着いて考察していけば正解は導けるが，設問数が少ないため，1問の配点が大きい。ケアレスミスに注意する必要がある。

［設問1］

排他制御の必要性を考えさせる設問であり，二重更新による不具合（正しい値とならない）のことを理解できていればよい。複数のプロセスが同時並行的に動くと，処理のタイミングによっては正しい値とならないことがある。共有データyの値が5で，プロセスp1がyの値を＋2，プロセスp2がyの値を－1するとき，プロセスp1の処理終了後に，プロセスp2が処理を行えば，共有データyの値は，5＋2－1＝6となる。これは，プロセスp1，p2の実行順序を逆にしても，同じ（5－1＋2＝6）である。しかし，図Aに示すように，実行順序を①～⑥の順番で行った場合，正しい結果が得られない。



図A　プロセスp1，p2による実行結果

最終的なyの値は4であり，6にならない。これを，二重更新による不具合（又は更新データの喪失）という。プロセスの実行順序にかかわらず，常に正しい値となるよう処理が行われなければならない。つまり，p1，p2のプロセスを並行して実行する場合，p1→p2又はp2→p1の順番で直列に実行した，どちらかの実行結果と同じになるということを，直列化可能という。直列化可能であるために排他制御を行う。なお，この問題ではp1→p2とp2→p1の実行結果が6となるが，p1→p2とp2→p1の実行結果が同じにならない場合もある。例えば，p1が共有データyの値を2倍し，p2が共有データyの値を1減少させる場合，p1→p2では2y－1，p2→p1では2（y－1）で同じ結果にはならない。

また，図Aにおいて，プロセスp2の書込みの後に，プロセスp1が書込みを行えば，最終的なyの値が7になることも想定できる。すなわち，二重更新にy＋2かy－1の処理のどちらかが失われることになる。y＋2が失われたときは，y－1だけの処理が有効となってy－1＝5－1＝4という実行結果となり，y－1が失われたときは，y＋2だけの処理が有効となってy＋2＝5＋2＝7という実行結果となる。

以上から，共有データyが取り得る値は，正しい値の6，及び誤った値の4，7であるため，取り得ない値は5であり，正解は（イ）となる。

［設問2］

排他制御の必要性は，設問1で確認したように，共有データ（資源）の読込みを同時並行的に行うプロセスに対して，同時には許さないことにある。つまり，プロセスp1か，プロセスp2のどちらかの処理が完全に終了した後で行えば問題ないことになる。そこで，プロセスp1が先に共有データyの読込みを行った場合，プロセスp2からの読込み要求があってもブロックして，排除する。これを排他制御のロックというが，その役割を担った関数がℓ（共有データの確保）である。また，処理が終了した後は，アンロックの役割を担った関数がu（共有データの解放）である。なお，同期変数sの役割は，確保状態なのか，それとも解放状態なのかを表すことである。

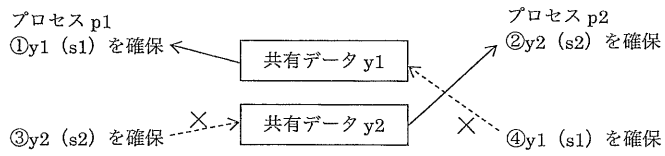
つまり，関数ℓの操作内容（空欄a）は，「sの状態が解放状態ならば確保状態にし，確保状態（他が使用中）ならば解放状態（使用可能）になるまで待ってから確保状態にする」（ア）が正解となる。いずれにしても，確保状態にするということが理解できていれば，（ア）か（イ）に絞られる。（イ）は，「確保状態ならば何もしない」とあるが，共有データの読込みを必要としているわけなので，通常は待ち状態に入る。何もしないということは，処理が進まないことになるので，誤りである。

また，関数uの操作内容（空欄b）は，「sの状態が確保状態ならば解放状態にし，解放状態ならば何もしない」（オ）が正解となる。処理が終了した後に呼び出される関数なので，共有データ（資源）を解放状態にし，他が使用できるようにするということに気が付けば，（オ）か（カ）に絞られる。しかし，（カ）は「解放状態になるまで待ち」と記述されており，解放状態にするという機能が含まれていないので，誤りである。設問文に，「①関数ℓの呼出し，②計算処理，③関数uの呼出し」の順番で処理を実行すると記述されているので，今使用しているプロセスが解放状態に戻すことになる。

［設問3］

デッドロックとは，排他制御によって共有データ（資源）がロック（確保状態で他が使用できない）されているため，お互いが待ち状態に陥り，処理が進まない状態をいう。

デッドロックの例を図Bに示す。（×は確保できないことを示す）



図B　デッドロックの例

①～④の順番で実行すると，プロセスp1は③でy2を確保しようとするが，y2はプロセスp2（②）によって既に確保されているので，解放されるまで待ち状態となる。一方，プロセスp2は④でy1を確保しようとするが，y1はプロセスp1（①）によって既に確保されているので，解放されるまで待ち状態となる。つまり，お互いが待ち状態に陥り，処理が進まない状態となる。これがデッドロックである。デッドロックは，この場合，二つの共有データ（資源）に対して，互いに逆の順番で共有データを確保しようとした結果として発生する。俗称ではあるが，一方の資源を確保し，互いに他の資源を確保しようとすることを矢印の線などで描くと，背中にたすきを掛けたように見えることから，“たすき掛け”のデッドロックということもある。デッドロックを防ぐには，確保する資源の順番を固定にすればよい。この例では，y1→y2からy2→y1のどちらかに固定すればよい。

設問では，プロセスp1が，①y1の確保，②y2の確保，③y2の解放，④y1の解放の順序で実行される場合についてが問われている。図Bで示した内容をイメージしながら，選択肢を吟味していくと，次のようになる。

ア：p1…① y1 確保 → ② y2 確保 → ③ y2 解放 → ④ y1 解放
　　p2…①’ y1 確保 → ②’ y1 解放 → ③’ y2 確保 → ④’ y2 解放

プロセスp1が先に「①y1の確保」を行った場合，y1が解放されるまで，プロセスp2はy1の確保（①’）ができない。プロセスp1が「④y1の解放」を行えば，プロセスp1は全ての処理を終えているので，プロセスp2の①’～④’までの処理は全て可能である。

また，プロセスp2が先に「①’ y1の確保」を行った場合は，「②’ y1の解放」を待ってから，プロセスp1が「①y1の確保」を行うことになる。このタイミングで，プロセスp2は先に，「③’y2の確保，④’y2の解放」を行うので，プロセスp1の②，③の処理は可能である。したがって，デッドロックとはならない。

イ：p1…① y1 確保 → ② y2 確保 → ③ y2 解放 → ④ y1 解放
　　p2…①’ y1 確保 → ②’ y2 確保 → ③’ y2 解放 → ④’ y1 解放

プロセスp1が先に「①y1の確保」を行った場合，y1が解放されるまで，プロセスp2はy1の確保ができない。（ア）と同様，プロセスp1が，「④y1の解放」を行った後，プロセスp1は全ての処理を終えているので，プロセスp2の①’～④’までの処理は全て可能である。

また，プロセスp2が先に「①’ y1の確保」を行った場合は，「④’ y1の解放」を待ってから，プロセスp1が「①y1の確保」を行うことになる。プロセスp2が「④’y1の解放」を行うのは全ての処理が終了した後なので，その後のプロセスp1の①～④までの処理は可能である。したがって，デッドロックとはならない。

ウ：p1…① y1 確保 → ② y2 確保 → ③ y2 解放 → ④ y1 解放
　　p2…①’ y2 確保 → ②’ y1 確保 → ③’ y1 解放 → ④’ y2 解放

プロセスp1が先に「①y1の確保」を行い，その直後に，プロセスp2が「①’y2の確保」を行ったとすると，プロセスp1は，「②y2の確保」ができない。

また，プロセスp2も，プロセスp1がy1を解放するまで，「②’y1の確保」ができない。したがって，デッドロックに陥る可能性がある。

エ：p1…① y1 確保 → ② y2 確保 → ③ y2 解放 → ④ y1 解放
　　p2…①’ y2 確保 → ②’ y2 解放 → ③’ y1 確保 → ④’ y1 解放

プロセスp1が先に「①y1の確保」を行った場合，y1が解放されるまで，プロセスp2はy1の確保ができない。しかし，プロセスp2が最初に行うのはy2の確保であり，①’及び②’の実行は可能である。y2の確保，解放はプロセスp1（②，③）も行うが，どちらが先に行っても，処理は可能である。プロセスp2は，プロセスp1のy1の解放を待って，③’，④’を実行できる。

また，プロセスp2が先に，①’，②’を実行した場合，y2の解放を待って，プロセスp1は②，③の実行に入ることになる。しかし，その前に①が実行されており，②，③を行うのは，①’，②’の後であるが，可能である。プロセスp2が③’，④’を行うのは，④の後であるが，可能である。デッドロックとはならない。

以上のことから，正解は（ウ）であるが，デッドロックのポイントは，図Bのように，同タイミングで，異なる資源を先に確保したときであるため，（ウ）か（エ）に絞られることに気が付けば，素早く正解を見いだせるだろう。