講　座

OS の基礎理論 (3)*

斎　藤　信　男**

4. スケジューリングの問題[1)]

現在の計算機システムが採用している多重処理システムに於ては，OS の定量的側面，即ち，その能率を左右するものは，与えられた資源を如何に有効に割り当てて，ユーザの仕事をやり遂げるかという点である．

これは，言い換えれば，与えられた資源をどのような方式で多重化すれば，全体の能率が向上するかということになる．この様な問題を抽象化して記述するためのモデルを設定し，能率を向上させるために用いるべき割り当てのアルゴリズムの追及や，それらの解析などを行うことを，スケジューリングの問題 (scheduling problem) という．

ユーザの仕事の遂行という点から考えると，最も重要な資源は，プロセサ (processor) である．もちろん，ここでいうプロセサは，いわゆる中央処理装置だけでなく，I/O プロセサ，通信制御プロセサなどを指していてもよい．この場合，ユーザの仕事は，プロセサをある時間だけ専有すれば終了するものと考える．プロセサの多重化を如何に行うかという点に注意したモデルとして，一つは，ユーザの仕事のステップの順序と，それぞれのステップで必要なプロセサの時間とがあらかじめ決められているシステムを対象としたものであり，これを，特に，**決定性モデル** (deterministic model)[1)] という．これに対して，オン・ライン・システムのような高度な多重処理システムに於ては，ユーザの仕事の要求の到着時間，あるいは，ユーザの仕事に必要なプロセサの時間などは確率的にしか決らないが，このようなシステムを対象としたモデルを，**確率モデル** (probability model)[1)] という．

ユーザの仕事の遂行にとってもう1つ重要な資源は，主記憶装置 (main memory) である．一つのプログラムにとっては，主記憶装置が割り当てられて，初めて，プロセサでの処理が可能となるのであるから，最近の大型機の OS のように高度な多重処理システムを管理する場合には，主記憶装置の多重化は，全体の能率にとって重要な要因となってくる．主記憶装置の多重化を能率良く行うためには，従来から，ページング方式が提案されていたが，最近の大型機では仮想記憶方式を採用する傾向が強く，ページングの管理は特に重要な問題となっている．

この章では，プロセサの多重化に関して，決定性モデル，確率モデルを概説し，また，主記憶装置の多重化に関して，ページング方式のモデルについて概説する．

4.1 決定性モデル

ここでは，複数個のジョブが，複数個のプロセサによって実行される時のスケジューリングの問題を考える．このとき，各ジョブは，複数個のタスクから成り立っており，そのタスク間の相互順序関係 <*** と，各タスクに要するプロセサの時間(タスクの実行時間)が，あらかじめ決められているものとする．タスクを節点で，その間の相互順序関係 < を弧であらわすと，このようなジョブの構成は，**Fig. 4.1** に示すような有向グラフ（**タスク・グラフ**）で表わすことができる．Fig. 4.1 に示したジョブに於ては，T_1 から T_5 までのタスクが存在し，たとえば，T_3 は，T_1 と T_2

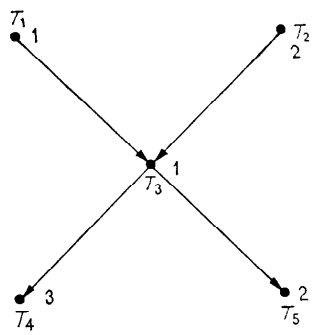


Fig. 4.1 task graph

* Fundamental Theory of Operating Systems by Nobuo SAITO (Institute of Informatics, University of Tsukuba).

** 筑波大学　電子・情報工学系

*** 半順序関係 (partial ordering) となる．

の両方が完了して初めて実行を開始できることをあらわしている．また，T_4 と T_5 は，T_3 が終了後，いつでも実行開始が可能であることを示している．一方，プロセサは，お互いに独立に動く同一のものが，複数個用意されているものとする．ここでは，主記憶装置の割り当ては考えないので，各タスクは，タスク・グラフで示された順序関係さえ満していれば，いつでもプロセサで実行が可能であると仮定する．

与えられたジョブと，与えられたプロセサとに対する**スケジュール** S とは，それぞれのプロセサを，どのような時間的経過で各タスクに割り当てるかを決めることである．このとき，与えられたジョブが全て終了するまでの時間を，**スケジュールの実行時間**といい，$t(S)$ であらわす．また，与えられたジョブとプロセサに対し，$t(S)$ が最小になるようにスケジュールしたものを，**最適スケジュール** (optimal schedule) と呼ぶ．スケジューリングの問題の最大の課題は，最適スケジュールを与えるアルゴリズムを求めることである．

なお，この場合，扱うべき対象は，有限のジョブとプロセサであるから，それらの全ての組み合せを考えて，数理計画法などの手法を用いれば，最適スケジュールが求められる．これに必要なステップ数は，出現するタスクの数の指数関数に比例する．それに対して，以下で求める最適スケジュールは，しらみつぶしに計算を行うのではなく，解析的な考察を用いて，必要なステップ数がタスクの数の多項式に比例するような手順で求められるアルゴリズムを追及するものとする．

4.1.1 2台のプロセサに対する最適スケジュール[2)]

任意のタスク・グラフで定義されるジョブに対する最適スケジュールを求めることは，一般に困難であるので，ここでは，幾つかの制限条件の下にスケジュールの問題を考えるものとする．即ち，同一のプロセサを2台用いること，各タスクの実行時間は同一であること，いったんプロセサで開始されたタスクは，途中で先取りが行われないことを仮定しておく．

定義 4.1

正整数を大きさの順に並べた減少列の2つ組 $N=(n_1, n_2, \cdots, n_t)$ と，$N'=(n_1', n_2', \cdots, n_{t'}')$ とに於て，

(1) $1\leqq i\leqq t$ なるある i に対して $n_i<n_i'$ が成り立ち，かつ，$1\leqq j\leqq(i-1)$ なる全ての j で $n_j=n_j'$ が成り立つ，

または，

(2) $t<t'$ で，かつ，$1\leqq j\leqq t$ なる全ての j で，$n_j=n_j'$ が成り立つ，

ならば $N\prec N'$ の関係にあるという．

定義 4.2（ラベル付け手続き）

n 個の節点を持ったタスク・グラフ G に対して，次に述べる手続きに従って，各節点 T に対し，$\{1, 2, \cdots, n\}$ の要素を選んで割り当てることを**ラベル付け**という．T に対するラベルを $\alpha(T)$ と表わす．

ステップ 1

G 上の節点 T の直後*の節点の集合を $S(T)$ であらわす．$S(T_0)$ が空であるような任意の節点 T_0 を1つ選び，$\alpha(T_0)=1$ とする．

ステップ 2

$k\leqq n$ なる k に対し，$1, 2, \cdots, k-1$ が既にラベルとして割り当てられていると仮定する．$S(T)$ の全ての節点に対し，既にラベル付けが終了しているような節点 T に対し，$\{\alpha(T')\mid T'\in S(T)\}$ の要素を大きさの順に並べた減少列を $N(T)$ とする．このような条件を満す節点の集合に対応する減少列 $N(T)$ の集合には，順序関係 $\prec$ に関して最小値が存在するが，これを T^* とし，$\alpha(T^*)=k$ とする．

ステップ 3

G内の節点に全てラベル付けが行われていれば，手続きを終了する．そうでなければ，ステップ2を繰り返す．

たとえば，**Fig. 4.2** (1) に示したタスク・グラフは，(2) に示したようにラベル付けが行われる．

定義 4.3 (A—スケジュール)

次のようにしてプロセサの割り当てを行うことを，**A—スケジュール**という．

ステップ 1

与えられたタスク・グラフに対し，定義4.2で述べた手続きを用いて，ラベル付けを行う．

ステップ 2

プロセサがあいたとき，直前**のタスクが全て実行を完了しており，かつ，未だプロセサを割り当てられていないタスクの間で，一番大きいラベルを持っているタスクを選び，プロセサに割り当てる．

たとえば，Fig. 4.2 (1) に示したタスク・グラフに対して，(2) で示すようなラベル付けが行われる

---

* T の出力弧（T から出ていく弧）を入力弧（入ってくる弧）とするような節点を，T の直後の節点という．

** ある節点 T への入力弧（入ってくる弧）を出力弧（出ていく弧）とするような節点を，T の直前の節点という．

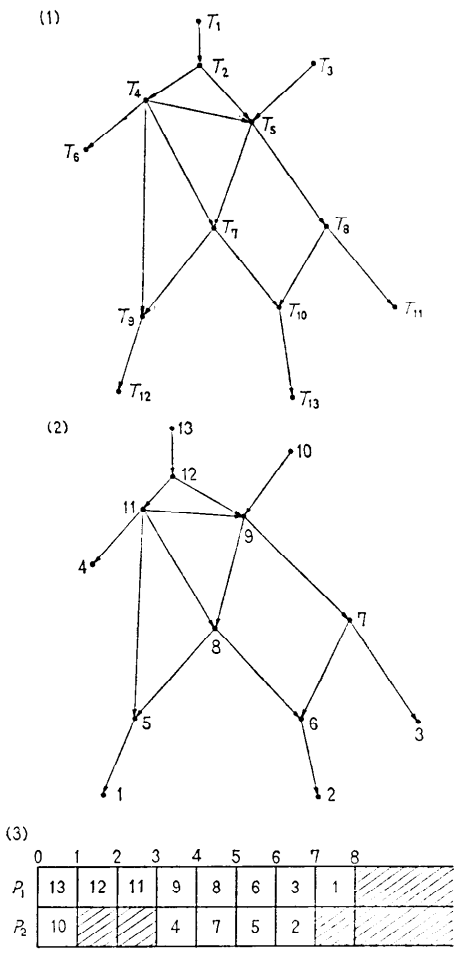


Fig. 4.2 A-schedule

が，プロセサ2台に対するA—スケジュールは，(3)に示ようになる．

A—スケジュールに関しては，次の定理が成り立つ．

定理 4.4

与えられた任意のタスク・グラフにおいて，タスクの所要サービス時間が同一で，かつ，プロセサの数が2のとき，A—スケジュールは，実行時間を最小とするスケジュールを与える．

定理4.4で示したように，各タスクの所要時間が同一で，かつ，プロセサの数が2の場合，A—スケジュールは，最適スケジュールになっていることがわかる．タスクの所要時間が同一でない場合，または，プロセサの数が2以上の場合は，最適とはならない．

4.1.2 木構造のタスク・グラフに対する最適スケジュール[3]

ジョブをあらわすタスク・グラフの形を木構造に制限した場合，プロセサの数を2以上にしても，最適スケジュールを与えるアルゴリズムが得られる．この場合も，各タスクの所要時間が同一であることは仮定しておく．

定義 4.5

木構造を持ったタスク・グラフGにおいて，木の根に相当するタスクを，終端タスクという．このとき，任意のタスクに対し，終端タスクからの最長径路（通過する節点の最大数）を，タスクのレベルといい，$L(T)$ であらわす．特に，終端タスクのレベルは，1とする．

定義 4.6 (B—スケジュール)

次のようにしてプロセサの割り当てを行うことを，**B—スケジュール**という．

プロセサがあいたとき，直前のタスクが全て実行を完了しており，かつ，未だプロセサを割り当てられていないタスクの間で，一番高いレベルを持つタスクを選び，プロセサに割り当てる．複数個のタスクが上の条件に合致している時は，その中から任意のものを一つ選ぶことにする．

B—スケジュールに関しては，次の定理が成り立つ．

定理 4.7

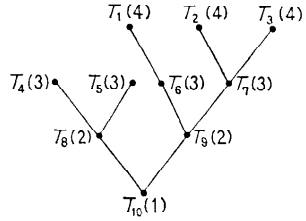
与えられたタスク・グラフが，木構造をもち，かつ，各タスクの所要サービス時間が同一のとき，B—スケジュールは，実行時間を最小とするスケジュールを与える．

定理4.7では，プロセサの数を任意個で良いとしている．なお，A—スケジュールを3個以上のプロセサの場合に拡張したとき，木構造を持ったタスク・グラフに適用すれば，B—スケジュールと等価なものとなる．

B—スケジュールの例を，**Fig. 4.3** (次頁参照) に示す．ここでは，プロセサの数は，3である．タスク・グラフの節点のカッコ内の数字は，定義4.5で与えられたレベルをあらわす．

4.1.3 スケジューリングにおける異常事態 (scheduling anomalies)[4]

あらかじめタスクを一定の順序（たとえば，優先順位）に並べたリストを用意し，プロセサがあいたとき，そのリストを一定方向にたどり，直前のタスクが全て実行を完了しており，かつ，リスト内で一番最初に見つかったタスクにプロセサを割り当てるというスケジューリングが考えられるが，これを，特に，**リス**



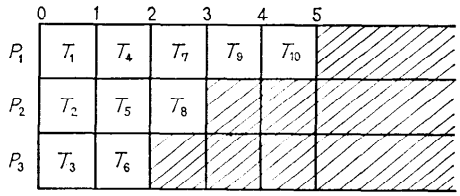
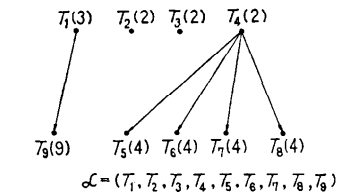


Fig. 4.3 B-schedule

ト・スケジューリング**という．A—スケジュールは，ラベルを降順に並べたリストを作れば，一つのリスト・スケジューリングとなる．

リスト・スケジューリングに於けるスケジュールの実行時間は，プロセサの数，各タスクの実行時間，タスク間の相互順序関係による制限，用いるリスト等によって決まってくるが，この場合，これらのパラメータを変更したとき，常識的に予想される結果と相反する結果が生ずることがある．これを，**スケジューリングにおける異常事態**という．たとえば，プロセサの数の増加，あるタスクの実行時間の縮小，相互順序関係による制限の撤廃等によって，かえって，スケジュールの実行時間が増加することがある．

いま，**Fig. 4.4** に示すようなタスク・グラフで与えられるジョブに対し，プロセサを3つ用い，リスト



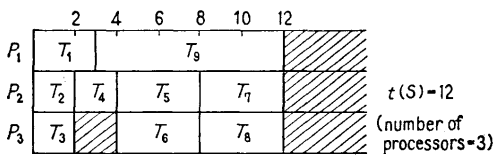


Fig. 4.4 List schedule (number of processors=3)

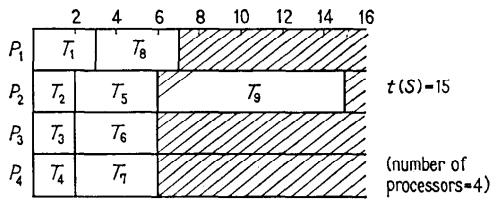


Fig. 4.5 Scheduling anomalies (increasing number of processors)

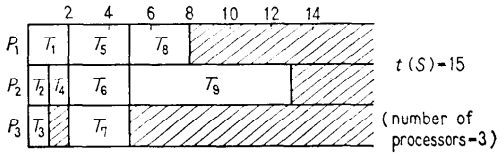


Fig. 4.6 Scheduling anomalies (decreasing task exeuction time)

$\mathcal{L}=(T_1, T_2, T_3, T_4, T_5, T_6, T_7, T_8, T_9)$ を用いて，リスト・スケジューリングを行った結果，スケジュールの実行時間は，12となる．

これに対し，他のパラメータは変更せずに，プロセサの数だけ4に増加して，リスト・スケジューリングを行うと，**Fig. 4.5** に示すような結果が得られ，予想に反して，スケジュールの実行時間は 15 にふえてしまう．

また，各タスクの実行時間を，Fig. 4.4 に与えられている値から1づつ減らして，次のようにする．

T_1: 2　T_4: 1　T_7: 3
T_2: 1　T_5: 3　T_8: 3
T_3: 1　T_6: 3　T_9: 8

他のパラメータは，Fig. 4.4 に与えたものと同一にして，リスト・スケジューリングを行うと，**Fig. 4.6** に示すような結果となり，予想に反して，スケジュールの実行時間は，13にふえてしまう．

また，Fig. 4.4 に示したタスク・グラフによる各タスク間の順序関係をゆるやかにして，**Fig. 4.7** (次頁参照) に示すようなタスク・グラフに変更する．他のパラメータは，Fig. 4.4 に与えたものと同一にして，リスト・スケジューリングを行うと，Fig. 4.7 に示すような結果となり，予想に反して，スケジュールの実行時間は16にふえてしまう．

スケジューリングの異常事態によって増加するスケジュールの実行時間の上限に対しては，次のような定理が成り立つ．

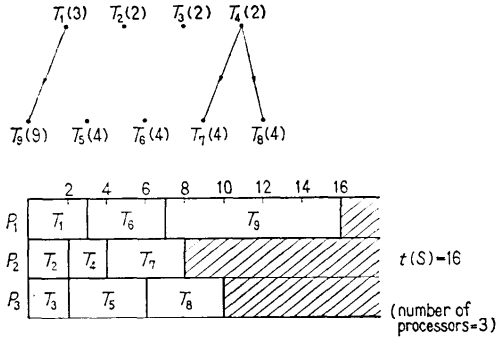


Fig. 4.7 Scheduling anomalies (relaxing the partial ordering)

定理 4.8

2つのリスト・スケジューリング S, S' のパラメータが，次のように与えられるとする．

S：タスクの集合：$\mathcal{T}=\{T_1, T_2, \cdots, T_n\}$

タスクの実行時間：$\tau_1, \tau_2, \cdots, \tau_n$

プロセサの数：m

タスク間の順序関係：$\prec$

タスクのリスト：$\mathcal{L}$

S'：タスクの集合：$\mathcal{T}=\{T_1, T_2, \cdots, T_n\}$

タスクの実行時間：$\tau_1', \tau_2', \cdots, \tau_n'$

$\{\forall i(1\leqq i\leqq n)\tau_i'\leqq\tau_i\}$

プロセサの数：m

タスク間の順序関係：$\prec$ に含まれる半順序関係 $\prec'$

タスクのリスト：$\mathcal{L}'$

このとき，それぞれのスケジュールの実行時間を t, t' とすると，

$$\frac{t'}{t}\leqq 1+\frac{m-1}{m'} \qquad (4\text{-}1)$$

が成り立つ．

たとえば，$m=3$, $m'=4$ とすると，$t'\leqq 3/2t$ となる．

4.2 確率モデル

オン・ライン・システムに於けるジョブの発生などは，確率的にしか決定できない．この様なジョブのスケジューリングを解析する為に，確率モデルを導入する．確率モデルで扱う現象は，決定性モデルに比べて，長期間にわたるものであり，得られる結果も巨視的なものである．従って，決定性モデルに比べて，計算機システムの実際の動作を忠実に表現できることになる．

ここでは，確率モデルとして，プロセサと，それに対するジョブの待ち行列を主として考える．待ち行列システムの理論は，OR の一分野として発展させられており，その成果を利用することができる[5]．待ち行列システムは，**Fig. 4.8** に示すように，プロセサに対応するサーバ (server)，それに付随する待ち行列，及びサービスを要求するジョブの発生源であるソース (source) の3つのサブ・システムから成り立つ．ソースから，新しいジョブが発生して，待ち行列に入ることを到着 (arrival) という．サーバは，ある処理方法に従って待ち行列内のジョブにサービスし，各ジョブの要求するサービス時間が満足されると，システムから離脱 (departure) する．

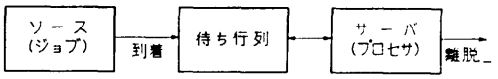


Fig. 4.8 Quene system

待ち行列システムは，一般に，ジョブの到着の時間間隔の確率分布，ジョブの要求するサービス時間の確率分布，及び待ち行列の処理方式，即ち，待ち行列内のどのジョブを選択してサービスを開始するのか，及び，サービスを開始したら，どれだけの時間のサービスを継続するのかという要素から記述される．到着の時間間隔をあらわす確率変数を T とすると，T の分布関数を $A(x)=\text{Probability}\{T\leqq x\}$ で表わす．また，サービス時間をあらわす確率変数を S とすると，S の分布関数を，$B(x)=\text{Probability}\{S\leqq x\}$ で表わす．

一般に，確率モデルのジョブの到着過程として，**完全ランダム過程**（または，**ポアソン過程**）を用いる．これは，$A(x)$ が指数分布

$$A(x)=1-e^{-\lambda x} \qquad (4\text{-}2)$$

として与えられる．また，時間 t 内に n 個の到着がある確率は，ポアソン分布

$$P_n(t)=\frac{(\lambda t)^n}{n!}e^{-\lambda t} \qquad (4\text{-}3)$$

で与えられる．ここに，パラメータ λ は，**到着率**といわれ，単位時間内の平均到着数を示す．したがって，到着の時間間隔の平均値は $1/\lambda$ であり，また，間隔の標準偏差も $1/\lambda$ である．また，指数分布 (4-2) は，どの時点においても同じ式で $A(x)$ が表わされるという性質を持つ．即ち，ある時間以降である事象が生ずる確率が，それまでの経過に全然依存しないで決定される．これを，**非記憶性** (memory-less property)，または，**マルコフ性** (Markov property) と呼び，指数分布の特有な性質と考えられる．

サービス時間の分布 $B(x)$ としては，いろいろの分布関数が考えられるが，指数分布とすれば，扱いは簡単になる．

サービスを開始したら，どれだけの時間それを継続するのかという点に関しては，先取り方式 (preemptive) と，非先取り方式 (non—preemptive) とが考えられる．非先取り方式では，いったん，一つのジョブへのサービスが始まると，そのジョブの要求する時間だけ継続してサービスを行う方式であり，先取り方式では，任意の時点でサービスの中断を許す方式である．TSS などのシステムでは，先取り方式で行うのが一般的であろう．

待ち行列内のどのジョブを選択するのかという点に関しては，FIFO (first-in first-out)，LIFO (last-in first-out)，ラウンド・ロビン方式 (round robin discipline)，優先度方式 (priority discipline) などの方式がある．これらのうち，どのような方式をとるにしても，到着するジョブに関する幾つかの情報，たとえば，到着時間，要求するサービス時間，残りのサービス時間，外部で要請された優先度などを充分生かして行わなければならない．

この様な確率モデルを用いて計算機システムの解析をする場合，どんな量を用いて性能を記述するかという問題がある．たとえば，待ち行列システム内のジョブの個数，各ジョブがシステム内で費さねばならない時間 (elapsed time)，待ち行列内での待ち時間 (waiting time)，サーバがサービスを行っている仕事時間 (busy period) などの分布関数がわかれば，システムの性能を評価することができるであろう．これらの分布関数は，一般には，統計的に平衡状態あるいは定常状態になったときのものを求める方が容易である．

4.2.1 ランダム到着・指数分布サービス時間・単一プロセサの待ち行列システム (M/M/1 system)

はじめに，到着過程がランダム過程で，サービス時間が指数分布を持ち，プロセサが1個の場合の待ち行列システムを考えてみよう．この待ち行列システムを，M/M/1 と略記するが，ここで，M はマルコフ (Markov) の頭文字を示している．到着間隔 T の分布関数 $A(x)$，サービス時間 S の分布関数 $B(x)$ は，

$$A(x)=1-e^{-\lambda x} \quad (x\geq 0) \tag{4-4}$$

$$B(x)=1-e^{-\mu x} \quad (x\geq 0) \tag{4-5}$$

で与えられるものとする．

待ち行列の処理方式は，ジョブの選択順序が，あらかじめ与えられたジョブの実行時間に全く依存せずに決められるような方式，たとえば，FIFO, LIFO などの方式を用いることとする．

時刻 t におけるシステム内のジョブの個数を $X(t)$ とし，$X(t)$ が n となる確率を，

$$p_n(t)=\text{Probability}\,[X(t)=n]$$
$$t>0,\quad n=0,1,2,\cdots \tag{4-6}$$

とする．統計的平衡状態における p_n を求めるために，次の平衡状態方程式を解く．

$$\mu p_{n+1}-(\lambda+\mu)p_n+\lambda p_{n-1}=0 \quad (n>1) \tag{4-7}$$

$$\mu p_1-\lambda p_0=0 \tag{4-8}$$

いま，**トラフィック密度**として，

$$\rho=\frac{\text{1つのジョブの平均サービス時間}}{\text{相続くジョブの到着間隔}}=\frac{\lambda}{\mu} \tag{4-9}$$

と定義すると，(4-7)，(4-8) の解は，

$$p_n=(1-\rho)\rho^n \qquad n=0,1,2,\cdots \tag{4-10}$$

で与えられる．

(4-10) は，幾何分布を与えているが，平衡状態が存在するためには，$\rho<1$ の条件が必要である．これは，ちょうど，平衡状態において，プロセサは，次のジョブが到着する以前に，一つ以上のジョブの要求に応えられる状態になっており，プロセサは充分にその仕事を処理できると期待してよいことを示している．したがって，待ち行列の長さは，過度の長さに達しないと考えられる．実際，待ち行列内に N 個のジョブがいる確率は，ρ^{N+1} であり，トラフィック密度が小さいときは，長い行列ができるのは，極めて稀である．

(4-10) の幾何分布の平均 $\bar{n}$ と分散 $VAR(n)$ は，

$$\bar{n}=\frac{\rho}{1-\rho} \tag{4-11}$$

$$VAR(n)=\frac{\rho}{(1-\rho)^2} \tag{4-12}$$

で与えられる．また，(4-10) より，$p_0=1-\rho$ となるから，トラフィック密度 ρ は，プロセサがサービスに従事中の時間の割合を示す．

トラフィック密度の大きさに対するシステムの様子を，**Table 4.1** (次頁参照) に示す．

4.2.2 ランダム到着・一般的分布サービス時間・単一プロセサの待ち行列システム (M/G/1 system)

次に，サービス時間の分布を一般的な分布関数とし，他のパラメータは，M/M/1 と同じ条件の待ち行列システムを考える．ここで，G は，一般的 (General)

Table 4.1[5] System behavior for M/M/1

トラフィック密度 ρ	プロセサのあいている確率 $1-\rho$	待ち行列の平均の大きさ $\rho/(1-\rho)$	待ち行列に4つ以上ジョブがいる確率 ρ^5
0.1	0.9	0.111	0.00001
0.2	0.8	0.250	0.003
0.3	0.7	0.429	0.002
0.4	0.6	0.667	0.010
0.5	0.5	1.000	0.031
0.6	0.4	1.500	0.078
0.7	0.3	2.333	0.168
0.8	0.2	4.000	0.328
0.9	0.1	9.000	0.590

の頭文字を示している．時刻 t におけるシステム内のジョブの数を $X(t)$ とすると，M/G/1 では，$X(t)$ は，もはや，マルコフ過程になっていない．しかし，i 番目のジョブの実行が終了した直後の時刻を t_i とすると，$t_1, t_2, \cdots$ なる離散時点列を考えれば，$X(t)$ はマルコフ性を持ち，$X(t_i)$ $i=1,2,\cdots$ はマルコフ・チェインを作る．

マルコフ・チェインの遷移確率を，

$$\pi_{i,j}=\text{Probability}\{X_{k+1}=j \mid X_k=i\} \qquad k=1,2,\cdots \tag{4-13}$$

とすると，これは k に依存しない．平衡状態における X_i の分布は，

$$\pi_j=\sum_{i=0}^{\infty}\pi_{ij}\pi_i \qquad j=0,1,2,\cdots \tag{4-14}$$

なる平衡状態方程式の解として求まる．

$B(x)$ は，一般的な分布関数として与えられるから，トラフィック密度は，

$$\rho=\lambda E^*(s) \tag{4-15}$$

で定義される．また，サービス時間の変動係数 $C(s)$ は，

$$C^2(s)=\frac{VAR(s)}{E^2(s)}=\frac{E(s^2)^*}{E^2(s)}-1 \tag{4-16}$$

で定義されるが，これらを用いると，X_i の期待値 $\bar{n}$ は，

$$\bar{n}=\rho+\frac{\rho^2\{1+C^2(s)\}}{2(1-\rho)} \tag{4-17}$$

で与えられる．(4-17) の第1項は，プロセサ内におけるジョブの個数の平均値を与え，第2項は，待ち行列内のジョブの個数の平均値を与えている．また，$\bar{n}$，すなわち，ジョブの個数は，サービス時間のバラツキに比例しているので，バラツキが大きいと，待ち行列が長くなることを示している．サービス時間が指数分布に従うならば，$C^2(s)=1$ であり，

$$\bar{n}=\rho+\frac{\rho^2}{1-\rho}=\frac{\rho}{1-\rho} \tag{4-18}$$

となり，(4-11) と一致する．また，サービス時間が全て同一であれば，$C^2(s)=0$ であり，

$$\bar{n}=\rho+\frac{\rho^2}{2(1-\rho)}=\frac{\rho(1-\rho/2)}{1-\rho} \tag{4-19}$$

となる．ρ が0に近づけば，(4-19) は (4-18) に近づき，また，ρ が1に近づけば，(4-19) は，(4-18) の約半分に減少する．

4.2.3 待ち行列の処理方式[6]

前述した待ち行列の処理方式には，幾つかの方式が考えられたが，ここでは，それらを簡単に紹介し，各々の方式に対する平均待ち時間を求め，また，それぞれの特徴を述べる．

（1） **FIFO 方式** (first-in first-out discipline)

これは，到着するジョブを，到着順に，非先取り方式でサービスを与える方式である．

FIFO 方式では，システム内のジョブの平均の個数 $\bar{n}$ は，平均の待ち時間 W の時間間隔に到着するジョブの個数に等しくなると考えてよい．従って，

$$\bar{n}=\lambda W \tag{4-20}$$

が成り立つ．これは，**Little の公式**[7] と言われ，定常状態においては，待ち行列内のサービス方式や，到着の過程に依存しないで成立する．ランダム到着過程で，サービス時間が一般的分布のとき，Little の公式と M/G/1 における $\bar{n}$ の表現 (4-17) を用いれば，FIFO 方式に対する平均待ち時間は，

$$W=E(s)\left[1+\frac{\rho\{1+C^2(s)\}}{2(1-\rho)}\right] \tag{4-21}$$

で与えられる．また，待ち行列内での平均待ち時間 V は，(4-21) から明らかなように，

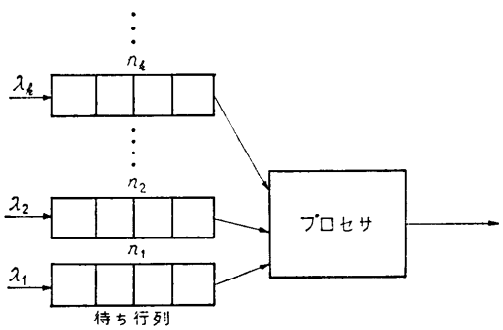


Fig. 4.9 Non-preemptive priority queue discipline

* $E(s)$ は，s の1次積率（期待値），$E(s^2)$ は，s の2次積率を表わす．

$$V=E(s)\cdot\frac{\rho[1+C^2(s)]}{2(1-\rho)} \qquad (4\text{-}22)$$

で与えられる.

(2) **非先取り・優先度方式** (non-preemptive priority discipline)

これは, 各ジョブにあらかじめ優先度が与えられ, **Fig. 4.9** に示すように, 優先度ごとに別々の待ち行列を設け, プロセサは, 優先度の高いものから非先取り方式でサービスを行う方式である. なお, 優先度は正整数で表わすが, 値の小さいもの程, 優先度が高いことにしておく. 同一の優先度内では, たとえば, FIFO 方式でサービスが行われる.

ジョブの到着は, 優先度ごとに独立なランダム過程に従うものとする. 優先度が k のジョブは, 到着率が $\lambda_k(k=1,2,\cdots)$ とする. また, サービス時間の分布は, 一般的なものとする. 平衡状態における k 番目の待ち行列に到着したジョブの平均待ち時間 W_k を考えると, W_k は, 優先度 k のジョブの待ち行列内の平均待ち時間 V_k と, 優先度 k のジョブの平均サービス時間 $E(s_k)$ との和で与えられるが, V_k は, 更に, 優先度1から k までのジョブの実行が終了するまでの平均時間 V_k', その間に到着した優先度1から $(k-1)$ までのジョブの実行が終了するまでの平均時間 V_k'', 及び, 注目しているジョブが到着したときに, プロセサでサービスを受けていたジョブの残り時間の平均値 $E(s_r)$ の和で与えられる. 従って,

$$\begin{aligned}W_k&=E(s_k)+V_k\\&=E(s_k)+E(s_r)+V_k'+V_k''\\&=E(s_k)+\frac{\sum_{i=1}^{\infty}\lambda_i E(s_i^2)}{2(1-\beta_k)(1-\beta_{k-1})}\end{aligned} \qquad (4\text{-}23)$$

となる. ここに, $E(s_i^2)$ は, 優先度 i のジョブのサービス時間の2次積率であり β_i は, $\beta_i=\sum_{j=1}^{i}\rho_j$ で与えられる. また, i 番目の待ち行列のトラフィック密度 ρ_i は, $\rho_i=\lambda_i E(s_i)$ で与えられる.

ジョブの要求するサービス時間が, ジョブ到着時にあらかじめわかっている場合は, その大きさに従って優先度をつけ, 上述した方式でサービスを行えば, 常にサービス時間が最短のジョブを実行していることになり, これを, 特に, **最短処理時間方式** (shortest-processing-time discipline) ということがある.

(3) **ラウンド・ロビン方式** (round-robin discipline)

これは, **Fig. 4.10** に示すように, プロセサは待ち行列の先頭からジョブを一つ取り出し, ある**刻み幅の時間** (quantum time) Q だけサービスを行ない, それだけでジョブの要求を満すことができなければ, もう一度待ち行列の最後尾に入れる方式である. これは, 時分割システムの最も簡単なスケジューリングであり, 先取り方式の1種とも考えられる. ラウンド・ロビン方式は, 文献8), 9), 10) などにより詳しく解析されている. このとき, Q だけのサービスを受けた後で, ジョブがシステムを離脱する確率を $(1-\sigma)$, 再び行列に入る確率を σ とする. また, ジョブの到着は, 到着率 λ のランダム過程に従うものとする. 各ジョブは, 刻み幅の整数倍の時間のサービスを受けることになるが, ジョブのサービス時間が iQ となる確率分布は,

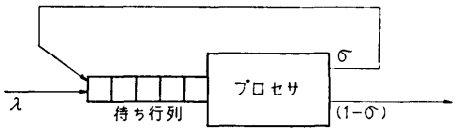


Fig. 4.10 Round robin discipline

$$g_i=\sigma^{i-1}(1-\sigma) \qquad (4\text{-}24)$$

なる幾何分布で与えられる. この分布の1次と2次の積率を求め, (4-17) を用いると, システム内のジョブの個数の平均値は,

$$\bar{n}=\rho+\frac{(1+\sigma)\rho^2}{2(1-\rho)} \qquad (4\text{-}25)$$

で与えられる. ただし, トラフィック密度 ρ は,

$$\rho=\frac{\lambda Q}{1-\sigma} \qquad (4\text{-}26)$$

で与えられる.

また, サービス時間を kQ 要求するジョブのシステム内の平均待ち時間 W_k は,

$$\begin{aligned}W_k&=W_1+\frac{(k-1)Q}{(1-\rho)}\\&\quad+Q\left[\lambda W_1+\sigma\bar{n}-\frac{\rho}{1-\rho}\right]\frac{1-\alpha^{k-1}}{1-\alpha}\end{aligned} \quad (k\geq 1) \qquad (4\text{-}27)$$

で与えられる. ここに,

$$\alpha=\lambda Q+\sigma \qquad (4\text{-}28)$$

$$W_1=\frac{1-\rho}{2}Q+\bar{n}Q \qquad (4\text{-}29)$$

である.

ラウンド・ロビン方式において, $Q/(1-\sigma)$ を一定に保ったまま, 刻み幅 Q を減少させてゆくと, 各ジョブの待ち行列内に繰り返し入る回数は増大する. こ

のとき，待ち行列内のジョブの相対位置の影響は減少するであろう．Q を0に近づけた極限の状態を，**プロセサ共有システム** (processor-sharing system) といい，Kleinlock[9] によりその性質が詳しく解析されている．プロセサ共有システムでは，ある時点で n 個のジョブがシステム内に存在すると，各々のジョブの受けるサービスは，1個のジョブがプロセサを専有したときに受けるサービスの丁度 $1/n$ になる．

プロセサ共有システムにおいては，サービス時間 $t=kQ$ を要求するジョブのシステム内の平均待ち時間 $W(t)$ は，

$$W(t)=\frac{t}{1-\rho} \tag{4-30}$$

で与えられる．

(4) **多段フィードバック方式** (multi-level feedback discipline)

ジョブの要求するサービス時間があらかじめわかっていない時には，ラウンド・ロビン方式が適用可能であるが，サービス時間の短いジョブを優遇して速く処理を終了する様なスケジューリングを行いたいことがあり，Corbató 等[11]によって，最初に提案された．そのために，**Fig. 4.11** に示すように，多段の待ち行列を設けるが，これを，多段フィードバック方式[10]という．ジョブの到着は，到着率 λ のランダム過程に従うものとする．システムに到着したジョブは，第1段目の待ち行列（優先度1）に入れる．ジョブは，ある定められた刻み幅の時間 Q だけサービスを受け，もしもそれで要求するサービス時間が満されない場合は，一つ上の段の待ち行列の後に入れられる．これを次々と繰り返してゆくので，第 k 段目の待ち行列（優先度 k）に入っているジョブは，既に $(k-1)\times Q$ 時間だけのサービスを受けたことになる．プロセサは，優先度の高い行列から処理してゆくので，k 段目の待ち行列内のジョブがサービスを受け始めるときには，最初の $(k-1)$ 個の待ち行列は，空になっているはずである．この方式では，システム内に滞在する時間が短いジョブから先に処理が終了してゆくことになるから，**最短滞在時間方式** (shortest-elapsed-time discipline) とも呼ばれる．

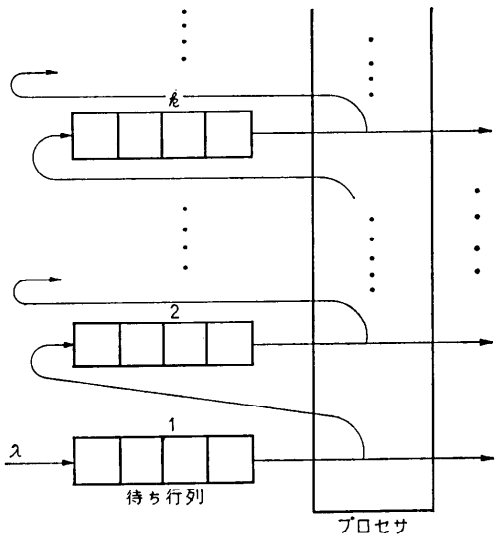


Fig. 4.11 Multi-level feedback quene discipline

ジョブが iQ のサービス時間を要求する確率を g_i，その分布関数を $G(k)=\sum_{i=1}^{k} g_i$ とする．kQ のサービス時間を要求するジョブに対するシステム内の平均待ち時間 W_k は，そのようなジョブが待ち行列で待たされる平均待ち時間 V_k と，kQ との和で与えられる．V_k は，更に，対象としているジョブが到着したときに実行していたジョブの平均終了時間と，そのとき最初の k 段の待ち行列に入っていたジョブの平均実行時間の和 V_k'，及び，対象とするジョブが $(k-1)$ 段の待ち行列を上ってくる間に新しく到着したジョブが，各々，$(k-1)\times Q$ のサービス時間を受けて k 段目の待ち行列に上ってくるまでの平均時間 V_k'' との和で与えられる．定義に従い，V_k, W_k を求めると，次のようになる．

$$V_k=\frac{\lambda\left[E_k(s^2)+Q^2\sum_{i=k}^{\infty}\{1-G(i)\}\right]}{2\{1-\lambda E_{k-1}(s)\}\{1-\lambda E_k(s)\}}+\frac{\lambda E_{k-1}(s)}{1-\lambda E_{k-1}(s)}(k-1)Q \tag{4-31}$$

$$W_k=V_k+kQ=\frac{\lambda\left[E_k(s^2)+Q^2\sum_{i=k}^{\infty}\{1-G(i)\}\right]}{2\{1-\lambda E_{k-1}(s)\}\{1-\lambda E_k(s)\}}+\frac{(k-1)Q}{1-\lambda E_{k-1}(s)}+Q \tag{4-32}$$

ここに，$E_k(s), E_k(s^2)$ は，最大 k 段目まで上ってきたジョブが受けたサービス時間の期待値，及び2次積率を表わし，

$$E_k(s)=\sum_{i=1}^{k}(iQ)g_i+kQ\{1-G(k)\} \tag{4-33}$$

$$E_k(s^2)=\sum_{i=1}^{k}(iQ^2)g_i+(kQ)^2\{1-G(k)\} \tag{4-34}$$

で与えられる．

（5） **最短残処理時間方式** (shortest-remaining-processing-time discipline)

各ジョブの要求するサービス時間があらかじめわかっており，先取り方式を用いる場合，任意の時点において，各ジョブのサービス時間の要求の残りが最小なものをサービスするという方式が採用できる．これを，最短残処理時間方式[12]という．この方式は，全ての待ち行列のサービス方式の中で，平均の待ち行列の長さが最小になるという秀れた特徴を持つ．

各ジョブの要求するサービス時間は，刻み幅の時間 Q の整数倍であると仮定し，サービス時間の確率分布を（4）と同様に，$g_i, G(k)$ であらわす．**Fig. 4.12** に示すように，多段の待ち行列を設け，残処理時間が kQ のジョブは，k 段目（優先順位 k）の待ち行列に入れる．各段の待ち行列に最初に入ってくるジョブは，到着率 $\lambda_k(k=1,2,\cdots)$ のランダム過程に従うものとするが，λ をシステム全体の到着率とすれば，$\lambda_k=\lambda g_k(k=1,2,\cdots)$ となる．k 段目の待ち行列内のジョブは，最初の $(k-1)$ 個の待ち行列が空にならない限り，サービスを受けられない．また，k 段目において，刻み幅 Q のサービスを受けたならば，$(k-1)$ 段目の待ち行列に入り，直ちにサービスを開始して貰う．もし，自分よりも優先度の高いジョブが到着したならば，直ちに先取りされて，適当な待ち行列の先頭に退避させられる．

この方式における kQ のサービス時間を要求するジョブの平均待ち時間 W_k は，そのジョブがシステムに到着してから最初にサービスを開始されるまでの平均経過時間 W_k' と，それ以後，システム内に滞在している平均時間 W_k'' との和で与えられる．W_k' と W_k'' とを定義に従って計算すると，W_k は，

$$W_k=W_k'+W_k''$$
$$=\frac{\lambda E_k(s^2)}{2[1-\lambda H_k(s)][1-\lambda H_{k-1}(s)]}+\sum_{j=1}^{k}\frac{Q}{1-\lambda H_{j-1}(s)} \quad (4\text{-}35)$$

となる．ここに，$E_k(s^2)$ は，k 段目までの処理が終った時に受けたサービス時間に対する2次積率で，(4-34) で与えられる．また，

$$H_k(s)=\sum_{i=1}^{k}(iQ)g_i \quad (4\text{-}36)$$

である．

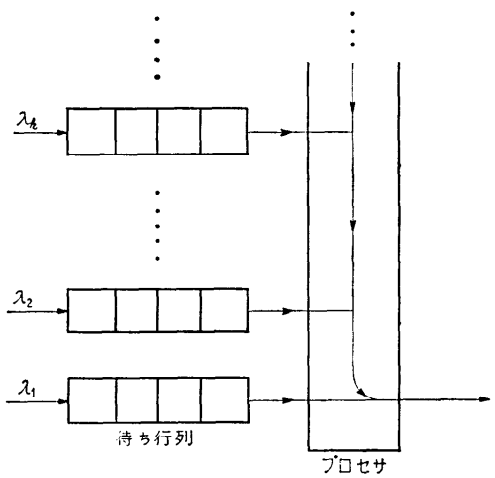


Fig. 4.12 Shortest-remaining-processing-time discipline

4.2.4 各処理方式の比較

最短残処理時間方式は，ジョブの要求時間の情報を事前に用い，また，任意の時点で先取りも許しているので，平均待ち時間を最小にする方式であることは，直観的にわかる．実際，一般的な到着過程と，サービス時間の分布に対して，平均待ち時間が最小になることが示される．これは，また，待ち行列の長さを最小にすることにもなる．ただし，この方式の欠点は，先取りが頻繁に起る可能性があり，先取りのための手間が増大する危険性がある．前述したモデルでは，そのようなオーバーヘッドは無限小と仮定して，考慮に入れていないが，先取りの手間による時間を無視しないようなモデルも作成できるであろう．

次に，FIFO 方式，ラウンド・ロビン方式，多段フィードバック方式を比較してみよう．サービスの要求時間の長いジョブと短いジョブとが混在する場合を考える．FIFO の方式と，ラウンド・ロビン方式とを比較すると，短いジョブの平均待ち時間は，ラウンド・ロビン方式の方がずっと短くなる．また，多段フィードバック方式では，システム内に存在するジョブの中で，処理時間の期待値が最小のものからサービスを与えていることになるから，短いジョブにとっては，ラウンド・ロビン方式よりも更に有利である．実際に，この3つの方式における平均待ち時間をグラフに示すと，**Fig. 4.13** （次頁参照）のようになる．

一方，サービス時間の分布との関連について考える．

例えば，サービス時間が一定 $(C(s)=0)$ のジョブだ

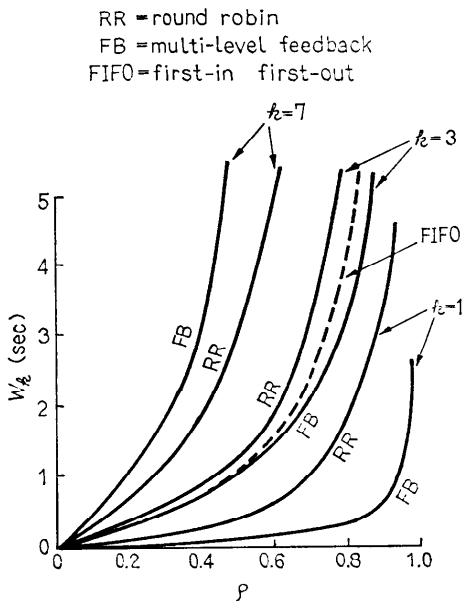


Fig. 4.13[10] Comparison of PR, FB, FIFO

けを考えると，FIFO 方式では，そのサービス時間だけ経過すると，少く共一つのジョブは完了するが，ラウンド・ロビン方式では，全てのジョブが終了するまで，一つもジョブが完了しない．一般に，サービス時間の変動が少い程，平均待ち時間に関して，FIFO 方式の方がラウンド・ロビン方式よりも有利である．また，このような場合，ラウンド・ロビン方式は，多段フィードバック方式よりも有利であることが予想される．

4.3 主記憶の管理のモデル—ページング方式のモデル[13]

主記憶の管理とその割り当ては，いろいろの方式が考えられるが，ここでは，最も進んだ方式と考えられるページング方式のみを対象とする．即ち，主記憶は，ある単位の大きさ（ページ・サイズ）に分割されており，割り当ては，ページを単位にして行われる．また，1つのプログラムの実行に際しては，必ずしも物理的に連続したページを割り当てる必要はないものとする．

4.3.1 ページの置き換え (page replacement)

あるプログラムの用いるページを，集合 $N=\{1,2,\cdots,n\}$ で表わす．1つのプログラムの動的な様子は，そのプログラムが実行中に参照するページの列である**参照列** (reference string)

$$\omega=r_1r_2\cdots r_k\cdots \qquad (r_k\in N, k\geq 1) \qquad (4\text{-}37)$$

で表わされる．一般に，システムの主記憶に用意されているページの数 m は，$1\leqq m\leqq n$ の関係にあると仮定する．プログラムの実行が行われているある時点で，参照すべき m 個以下のページを含んでいる N の部分集合 S ($|S|\leq m$) を，その時点での記憶状態といい，そのような部分集合全ての集合を，M_m で表わす．主記憶内に物理的に用意されているページの数は，プログラムで必要とする数よりも少いから，実行中のいずれかの時点で，ページの置き換えをしなければならない．プログラムの動作の履歴を覚えておくために，制御状態 $q\in Q$ を用い，対 (s,q) でシステムの状態を表わすこととする．システムの状態が (s,q) のときに，ページ i が参照されると，**置き換えアルゴリズム(ページング・アルゴリズム)** は，新しいシステムの状態 (s',q') を決める．即ち，ある写像

$$g:\ M_m\times Q\times N\to M_m\times Q$$
$$g(S,q,i)=(S',q')\quad i\in S' \qquad (4\text{-}38)$$

に従って，新しい記憶状態と制御状態とが決められる．これは，置き換えアルゴリズム A が，3つ組 $A=(Q,q_0,g)$ (ただし，q_0 は初期制御状態) で定義されることを示している．

実際に必要になった時点で初めてページの置き換えを行う方式を，**デマンド・ページング方式** (demand paging discipline) というが，A がデマンド・ページング方式における置き換えアルゴリズムのときに，$(S',q')=g(S,q,i)$ であるならば，S' は，次の性質を満していなければならない．

(1) $i\in S$ ならば，$S'=S$

(2) $i\notin S$ で，かつ，$|S|<m$ ならば，$S'=S\cup\{i\}$

(3) $i\notin S$ で，かつ，$|S|=m$ ならば，A があるページ $j\in S$ を選択して，$S'=(S-\{j\})\cup\{i\}$ とする．(ページ i がページ j と置き換る．)

非デマンド・ページング方式の置き換えアルゴリズム A に対しては，どんな参照列，ページの数に対しても，A よりも**ページ・フォールト** (page fault) の数が増加しないデマンド・ページング方式の置き換えアルゴリズム A' が存在することが言える[14]．従って，デマンド・ページング方式を解析しておけばよいことになる．

上述の (3) の場合に，どのページ j を選択するかによって，次のような置き換えアルゴリズムが考えられる．

(A1) LRU (least recently used) アルゴリズム

最後に参照された時点が，現在から一番離れてい

るページを置き換える方式.

(A2) ベラディ・アルゴリズム (Belady algorithm)[15]

これから参照される時点が，現在から一番離れているページを置き換える方式.

(A3) LFU (least frequenthy used) アルゴリズム

現在までに参照された頻度が一番少ないページを置き換える方式.

(A4) FIFO アルゴリズム

現在の時点で，主記憶の中に存在していた時間が最長のページを置き換える方式.

(A5) LIFO アルゴリズム

現在の時点で，主記憶の中に存在していた時間が最短のページを置き換える方式.

ここで，LRU, LFU, FIFO, LIFO は，プログラムの動作の過去の情報のみを用いて，置き換えるべきページの選択をしているので，実現可能であるが，ベラディ・アルゴリズムは，将来の動作の情報を必要とするので，実現不可能なアルゴリズムである.

4.3.2 最適置き換えアルゴリズム

システムに用意されているページの数を m，参照列を $\omega=r_1r_2\cdots r_T$，置き換えアルゴリズムを A とする．このアルゴリズムによって生ずるコストを，$C(A,m,\omega)$ で表わす．時刻 t における記憶状態 S_t は，$S_t=S_{t-1}+X_t-Y_t$ で表わされる．ここに，X_t は，時刻 t で新しく入ってくるページの集合，Y_t は置き換えられるページの集合であり，$X_t\cap Y_t=\phi$ である．デマンド・ページング方式では，

$$C(A,m,\omega)=\sum_{t=1}^{T}|X_t| \qquad (4\text{-}39)$$

で表わされると考えてよい.

任意の参照列に対して，コストを最小にするような置き換えアルゴリズムが最適アルゴリズムである.

デマンド・ページング方式では，(4-39) からわかるように，なるべくページの入れ換えを少くすればよいことになるから，ページ・フォールトの確率を最小にすればよいことになる.

ページの数を m，置き換えアルゴリズムを A とすると，ページフォールトの確率は，

$$f(A,m)=\Sigma_{\text{all }\omega}\,\text{Probability}\,[\omega](F(A,m,\omega)/|\omega|) \qquad (4\text{-}40)$$

で与えられる．ここに，Probability $[\omega]$ は，参照列 ω の生ずる確率，$F(A,m,\omega)$ は，ページ・フォールトの数，$|\omega|$ は，ω の長さを表わす.

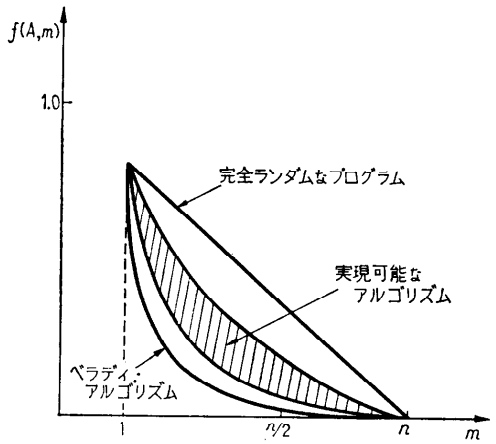


Fig. 4.14 Page foult rate

ページ m と，ページ・フォールトの確率との関係を，**Fig. 4.14** に示す．これからわかるように，実現可能なアルゴリズムに対する $f(A,m)$ は，m の大きさに大きく依存して変化するが，置き換えアルゴリズム A は，どれを用いてもあまり差がないことがわかる.

ページの数 m を増大した場合，ページ・フォールトの確率 $f(A,m)$ は，直観的には，常に低下しそうであるが，あるアルゴリズムに対しては，m の増加が，$f(A,m)$ の増大を招いてしまうような場合がある．Mattson 等[14]は，$f(A,m)$ の値が，m の増加に伴い，必らず減少するようなアルゴリズムのクラスを見つけ，このクラスを，**スタック・アルゴリズム** (stack algorighm) と呼んだ．即ち，ω を参照列，ページの数を m としたとき，参照列 ω を処理し終ったときの記憶状態を $S(A,m,\omega)$ とあらわすと，置き換えアルゴリズム A がスタック・アルゴリズムであるためには，

$$S(A,m,\omega)\subseteq S(A,m+1,\omega) \qquad (4\text{-}41)$$
$$(1\leqq m<n)$$

が成り立つことが必要である．即ち，m 個のページの内容は，常に $(m+1)$ 個のページに含まれているようなアルゴリズムのクラスであることを示している．たとえば，LRU アルゴリズムは，スタック・アルゴリズムであることが容易にわかる.

4.3.3 ワーキング・セット・モデル

多重プログラミング・システムにおいては，与えられた主記憶を幾つかの集団に分割して各プログラムに割り当てを行うが，どの様に割り当てればよいか，ど

の様なタイミングでページの置き換えを行えばよいかなどを決定するためには，各プログラムのふるまいを何らかの客観的な量で表現しておく必要がある．このような量をあらわす一つの視点は，**局所性** (locality) という性質である．一般に，一つのプログラムの実行経過をある時間間隔にわたって観察していると，そのプログラムが持つページを全て均等に参照するのではなく，その部分集合だけを特に頻繁に参照していることがわかる．この部分集合の要素は，ゆっくりと変化している．この局所性は，プログラムの一般的な構成法から考えると，充分に期待される性質である．たとえば，プログラムの実行のある時点を考えると，プログラム内の一つのモジュールを実行しているはずであり，それを構成するサブ・ルーチンあるいはデータだけを頻繁に参照するはずである．また，プログラムは，ループを使って構成されることが多いが，その場合，比較的小さいループであれば，それに属する命令の具合は，少い数のページ内に含まれてしまう事が多く，従ってある一部分のページを頻繁に参照することになる．

このような局所性をあらわすモデルとして，**ワーキング・セット・モデル** (working set model) があり，Denning 等[16),17)] によってその性質が詳しく究明されている．プログラムのワーキング・セットとは，直観的には，ページ・フォールト率から考えて，あるレベルの能率で実行されるために主記憶内にロードしておかなければならない最小のページの集合であると考える．

定義 4.9

与えられたプログラムの時刻 t におけるワーキング・セット $W(t, T)$ は，参照列 $r_{t-T+1}, \cdots, r_t$ の中に現われる全ての異なったページの集合である．$t<T$ であればは，$W(t, T)$ は，$r_1, r_2, \cdots, r_t$ 内の異なったページの集合となる．また，t が 0 以下であれば，$W(t, T)$ は空集合とする．パラメータ T を，ワーキング・セットの窓の大きさ (window size) という．また，$W(t, T)$ に含まれるページの数を，そのワーキング・セットの大きさ (working set size) といい，$w(t, T)$ であらわす．

一つのプログラムを実行するために，そのワーキング・セットを主記憶にロードしておかなければならないという原則で主記憶の管理を行えば，最適な管理をしていることになる．

参照列における最初の k 個の参照に対するワーキング・セットの大きさの平均値は，

$$S_k(T)=\frac{1}{k}\sum_{t=1}^{k} w(t, T) \tag{4-42}$$

で与えられるが，定常状態における平均値は，

$$S(T)=\lim_{k\to\infty}\frac{1}{k}\sum_{t=1}^{k} w(t, T) \tag{4-43}$$

で与えられる．$S(T)$ に関しては，次の 3 つの性質が成り立つ．

(1) $1\leq S(T)\leq \min\{n, T\}$

(2) $S(T)\leq S(T+1)$ (非減少)

(3) $S(T+1)+S(T-1)\leqq 2S(T)$ (上に凸)

上の性質から，T と $S(T)$ とは **Fig. 4.15** で示す関係になる．

ワーキング・セットの窓の大きさを T とするとき，確率変数 $\Delta(t, T)$ を，

$$\Delta(t, T)=\begin{cases}1 & r_{t+1}\notin W(t, T)\\ 0 & \text{それ以外の場合}\end{cases} \tag{4-44}$$

と定義すると，定常状態におけるページ・フォールトの起る確率 $m(T)$ は，

$$m(T)=\lim_{k\to\infty}\frac{1}{k}\sum_{t=0}^{k-1}\Delta(t, T) \tag{4-45}$$

で与えられる．**Fig. 4.16** からわかるように，

$$w(t+1, T+1)=w(t, T)+\Delta(t, T) \tag{4-46}$$

が成り立つが，両辺を 1 から k まで加えて，k で割り，k を無限大にした極限値をとると，

$$S(T+1)-S(T)=m(T) \tag{4-47}$$

が成り立つ．これは，Fig. 4.15 に示すように，$S(T)$

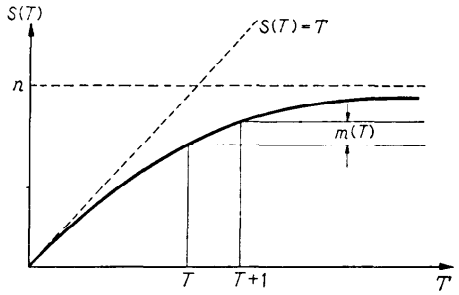


Fig. 4.15 Working set size

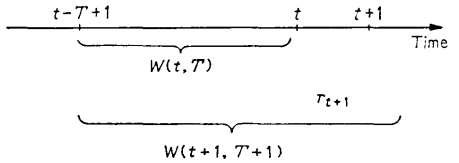


Fig. 4.16 Inclusion property of working set

のグラフの勾配が $m(T)$, 即ち, ページ・フォールト率で与えられることを示している.

なお, ワーキング・セットという考え方は, 種々の問題の解明の基礎的な概念として用いられており, 重要な役割を果している.

5. おわりに

3回にわたって, OS の解析, 設計などの基礎となりそおな理論を紹介してきた. この講座を終るにあたり, 次の2点を今後の問題として述べておく.

一つは, OS の扱うべき問題で, この様な基礎理論の展開が未だ不充分な分野が多く残されているという点である. この講座で扱かった話題でも, 同期基本命令の意味論, 並行処理プロセスの正当性の検証, 資源割り当ての問題の一般化, 資源割り当て問題における定量的側面の検討, 直列型待ち行列モデルの解析, プログラムの構造とワーキング・セットとの関係等, 今後追及されるべき問題が多々ある. また, この講座で扱わなかったが, プロテクション, セキュリティ, 信頼性, システム構成論, システム性能評価などの分野でも, 基礎理論の展開されることが望ましい.

もう一つは, これらの基礎理論と, 具体的な OS との相互関係である. 資源割り当てのアルゴリズムを具体的に埋め込んで本当によい OS が実現できるのか, 待ち行列モデルの結果がシステムの能率を向上させるための具体的なスケジューラのアルゴリズムの決定に役立つのか等といった疑問が生じてくる. 恐らく, Habermann のアルゴリズムを全ての資源割り当ての際に用いていたら, システムのオーバーヘッドは異常に増大するであろう. また, スケジューラのアルゴリズム, あるいは, プログラムの優先度の付け方は, ほとんど直観的にしか決められていないのが現状であろう. OS は, 元来, プログラムの複雑な集合であり, 泥くさいものであるが, その設計, 運用, 評価等において, 基礎的な理論の結果を反映させてゆく事が必要であり, そのための方法論も, また追及してゆかなければならない.

参 考 文 献

1) E. G. Coffman & P. J. Denning: *Operating System Theory*, Prentice-Hall, Englewood Cllifs, New Jersey (1973).
2) E. G. Coffman & R. L. Graham: Optimal Scheduling for Two-Processor Systems, Acta Informatica, Vol. 1, No. 3, pp. 200～213 (1972).
3) T. C. Hu: Parallel Sequencing and Assembly Line Problems, Operations Research, Vol. 9, No. 6, pp. 841～848 (1961).
4) R. L. Graham: Bounds for Certain Multiprocessing Anomalies, Bell System Technical J., pp. 1563～1581 (1966).
5) たとえば, D. R. Cox and W. L. Smith: *Queues*, Mathuen & Co. Ltd., London (1961). 磯野 修訳: 待ち行列, 日本評論社 (1966).
6) J. M. McKinney: A Survey of Analytical Time-Sharing Models, Computing Surveys, Vol. 1, No. 2, pp. 105～116 (1969).
7) J. D. C. Little: A Proof for the Queuing Formula $L=\lambda W$, Operations Research, Vol. 9, No. 3, pp. 383～387 (1961).
8) A. L. Sherr: An Analysis of Time-shared Computer Systems, M. I. T. Press, Cambridge (1967).
9) L. Kleinlock: Time-shared Systems: A Theoretical Treatment, JACM, Vol. 14, No. 2, pp. 242～261 (1967).
10) E. G. Coffman & L. Kleinlock: Feedback Queuing Models for Time-shared Systems, JACM, Vol. 15, No. 1, pp. 549～576 (1968).
11) F. J. Corbató et al.: An Experimental Time-Sharing System, Proc. AFIPS SJCC, Vol. 21, pp. 335～344 (1962).
12) L. W. Miller & L. E. Schrage: The Queue M/G/1 with Shortest Remaining Processing Time Discipline, Operations Research, Vol. 14, pp. 670～683 (1966).
13) P. J. Denning: Virtual Memory, Computing Surveys, Vol. 2, No. 3, pp. 153～189 (1970)
14) R. L. Matton et al.: Evaluation Techniques for Storage Hierarchies, IBM System J., Vol. 9, No. 2, pp. 78～117 (1970).
15) L. A. Belady: A Study of Replacement Algorithms for Virtual Storage Computers, IBM System J., Vol. 5, No. 2, pp. 78～101 (1966).
16) P. J. Denning: The Working Set Models for Program Behavior, CACM, Vol. 11, No. 5, pp. 323～333 (1968).
17) P. J. Denning & S. C. Schwartz: Properties of the Working Set Model, CACM, Vol. 15, No. 3, pp. 191～198 (1972).

(昭和49年10月23日受付)
(昭和49年11月11日再受付)