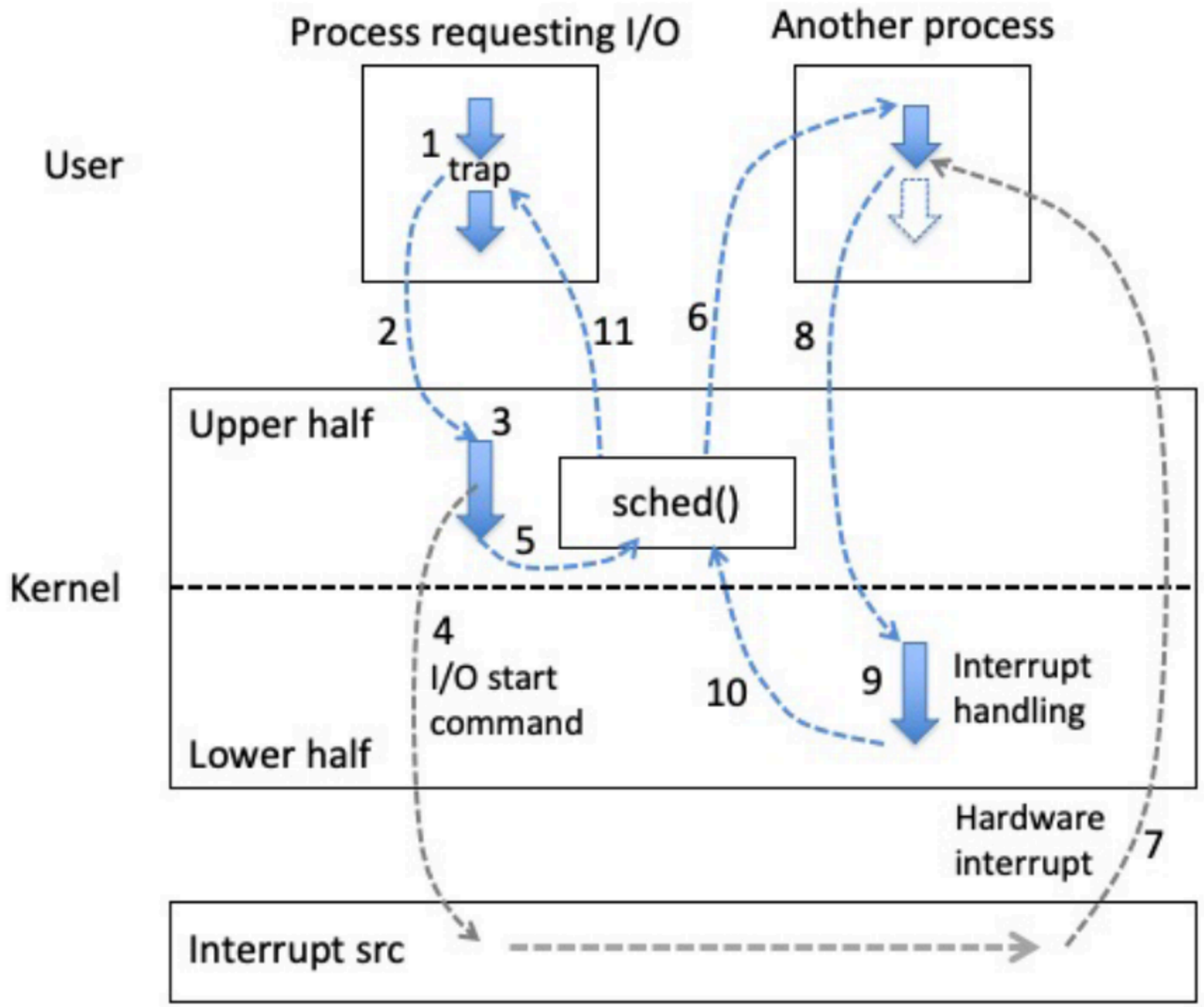
Scheduling

I233 Operating Systems

2.4 Scheduling

- 2.4.1 Introduction to Scheduling
  - Process Behavior
  - When to Schedule
  - Categories of Scheduling Algorithms
  - Scheduling Algorithm Goals
- 2.4.2 Scheduling in Batch Systems
  - First-Come, First-Served
  - Shortest Job First
  - Shortest Remaining Time Next
- 2.4.3 Scheduling in Interactive Systems
  - Round-Robin Scheduling
  - Priority Scheduling
  - Multiple Queues
  - Shortest Process Next
  - Guaranteed Scheduling
  - Lottery Scheduling
  - Fair-Share Scheduling
- 2.4.4 Scheduling in Real-Time Systems
- 2.4.5 Policy Versus Mechanism
- 2.4.6 Thread Scheduling

Location and behavior of the scheduler — Example on I/O block



Background

- CPU time is a scarce resource
  - With the advent of personal computers, the situation changed a bit. But still the importance of scheduling has not changed
- Focus on user experience
- Time and effort required for process switching (context switching)

Process Behavior

- Repeating calculations and input/output
- Compute-bound (CPU-bound) and I/O-bound.
  - As CPUs get faster, processes tend to get more I/O-bound

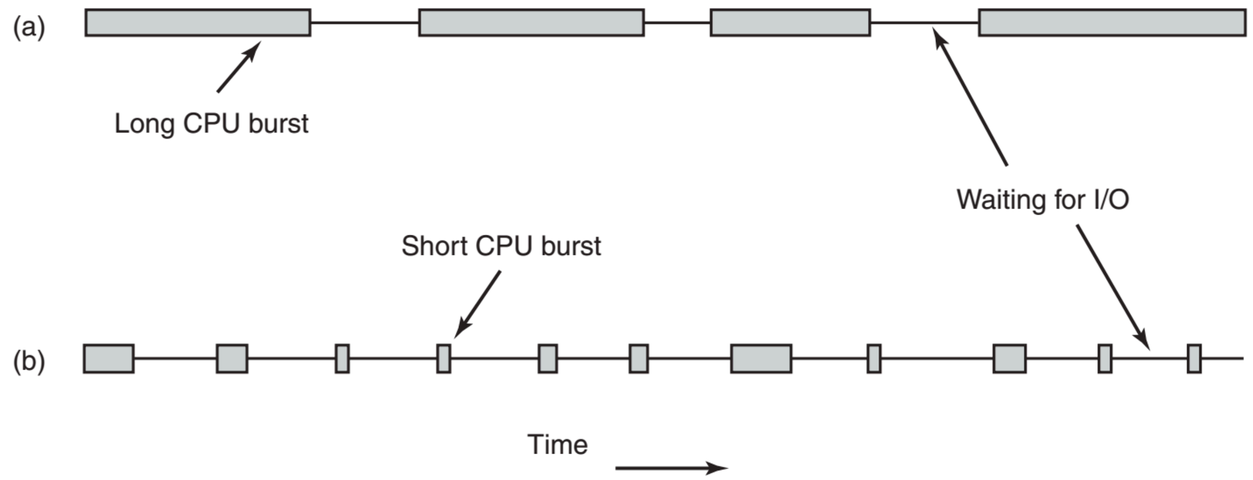


Figure 2-39. Bursts of CPU usage alternate with periods of waiting for I/O. (a) A CPU-bound process. (b) An I/O-bound process.

Source: Andrew S. Tanenbaum. “Modern Operating Systems”, 4th ed., global ed., pp.151.

When to Schedule

1. When a child process is created.
2. When a process exits.
3. When a process blocks on I/O, on a semaphore, or for some other reason.
4. When an I/O interrupt (notifying “completed its work”) occurs.
5. When a hardware clock interrupt occurs.

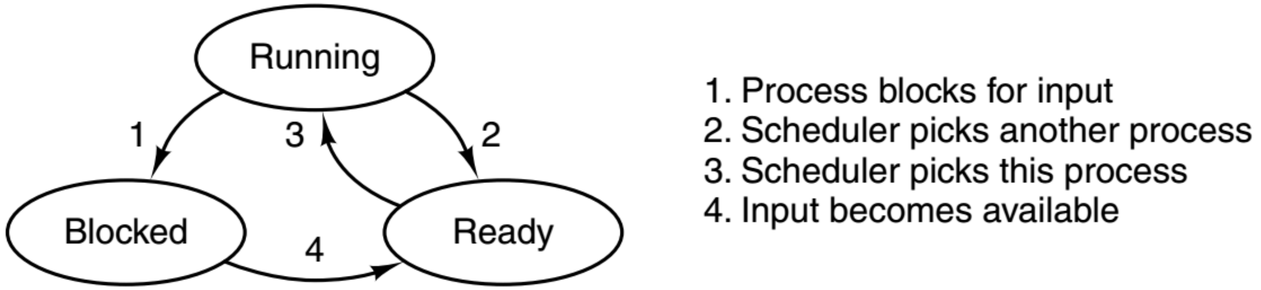


Figure 2-2. A process can be in running, blocked, or ready state. Transitions between these states are as shown.

Source: Andrew S. Tanenbaum. “Modern Operating Systems”, 4th ed., global ed., pp.93.

Nonpreemptive v.s. Preemptive scheduling

- Nonpreemptive scheduling algorithm
  - Picks a process to run and then just lets it run until it blocks (either on I/O or waiting for another process) or voluntarily releases the CPU.
- Preemptive scheduling algorithm
  - Picks a process and lets it run for a maximum of some fixed time (“quantum”).
  - With the clock interrupt, the control of the CPU is returned to the scheduler.

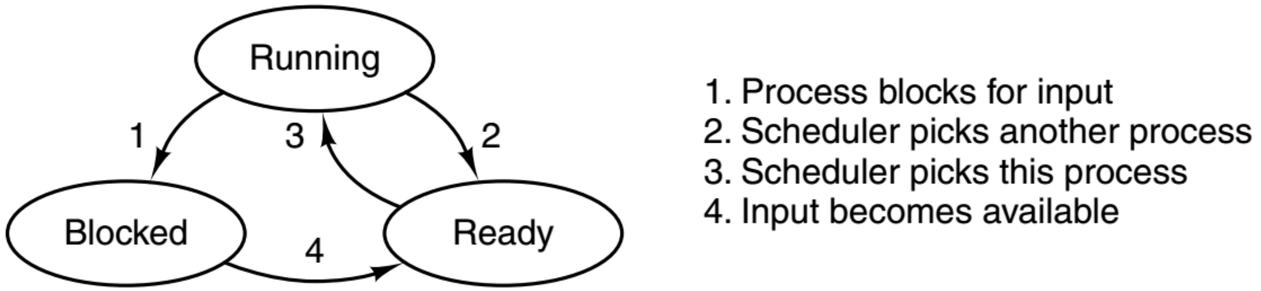


Figure 2-2. A process can be in running, blocked, or ready state. Transitions between these states are as shown.

Source: Andrew S. Tanenbaum. “Modern Operating Systems”, 4th ed., global ed., pp.93.

Categories of Scheduling Algorithms

- In different environments different scheduling algorithms are needed.
- Three environments worth distinguishing:
  - Batch
  - Interactive
  - Real time
- Some systems need to support multiple types of environments simultaneously
- A program with phases (each phase may show different behavior)

Concerns on scheduling algorithm

- Throughput
  - The number of jobs per hour that the system completes
- Latency
  - Turnaround time
    - The statistically average time from the moment that a batch job is submitted until the moment it is completed
  - Response time
    - The time between issuing a command and getting the result.
- Fairness

Scheduling Algorithm Goals (1)

- All systems (common)
  - Fairness
    - Giving each process a fair share of the CPU
  - Policy enforcement
    - Seeing that stated policy is carried out
  - Balance of each parts of the system
    - Keeping all parts of the system busy

Scheduling Algorithm Goals (2)

- Batch systems
  - Throughput — maximize jobs per hour
  - Turnaround time — minimize time between submission and termination
  - CPU utilization — keep the CPU busy all the time
- Interactive systems
  - Response time — respond to requests quickly
  - Proportionality — meet users' expectations
- Real-time systems
  - Meeting deadlines — avoid losing data
  - Predictability — avoid quality degradation in multimedia systems

Scheduling in Batch Systems

- 3 typical scheduling algorithms for batch systems
  - First-Come, First-Served
  - Shortest Job First
  - Shortest Remaining Time Next

First-Come, First-Served

- Nonpreemptive
- Processes are assigned the CPU in the order they request it.
- A single queue of ready processes.
- Pros.
  - Simple (easy to understand and equally easy to program)
  - Fair in the same sense
- Cons.
  - Differences in treatment of CPU-bound and I/O-bound processes

    e.g. Suppose there is one compute-bound process that runs for 1 sec at a time and many I/O-bound processes that use little CPU time but each have to perform 1000 disk reads to complete. The compute-bound process runs for 1 sec, then it reads a disk block. All the I/O processes now run and start disk reads. When the compute-bound process gets its disk block, it runs for another 1 sec, followed by all the I/O-bound processes in quick succession.

    The net result is that each I/O-bound process gets to read 1 block per second and will take 1000 sec to finish. With a scheduling algorithm that preempted the compute-bound process every 10 msec, the I/O-bound processes would finish in 10 sec instead of 1000 sec, and without slowing down the compute-bound process very much.

Shortest Job First

- Assuming the run times are known in advance
- Nonpreemptive
- The scheduler picks the shortest (run times) job first.
- Properties:
  - The average wait time (turnaround time) is short.

Figure 2-41. An example of shortest-job-first scheduling. (a) Running four jobs in the original order. (b) Running them in shortest job first order.

Source: Andrew S. Tanenbaum. “Modern Operating Systems”, 4th ed., global ed., pp.157.

Shortest Remaining Time Next

- A preemptive version of shortest job first
- When a new job arrives, its total time is compared to the current process' remaining time. If the new job needs less time to finish than the current process, the current process is suspended and the new job started.
- This scheme allows new short jobs to get good service.

Scheduling in Interactive Systems

- Round-Robin Scheduling
- Priority Scheduling
- Multiple Queues
- Shortest Process Next
- Guaranteed (promise) Scheduling
- Lottery (random) Scheduling
- Fair-Share Scheduling

Round-Robin Scheduling

- Maintaining a list (single queue) of runnable processes
- Each process is assigned a time interval, called its “quantum”, during which it is allowed to run.
- If the process is still running at the end of the quantum, the CPU is preempted and given to another process. Round robin is easy to implement. All the scheduler needs to do is maintain a list of runnable processes.

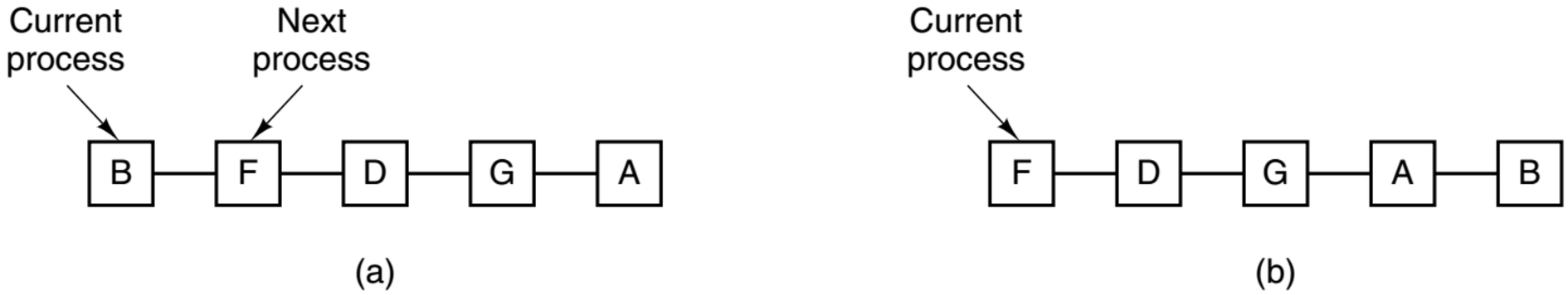


Figure 2-42. Round-robin scheduling. (a) The list of runnable processes. (b) The list of runnable processes after *B* uses up its quantum.

Source: Andrew S. Tanenbaum. “Modern Operating Systems”, 4th ed., global ed., pp.159.

Concerns about round-robin

- How is the value of Quantum determined?
- Should all processes be treated equally?
- The process that became executable on completion of I/O will be listed at what position in the list (top? last?) of the list?

Analysis of round-robin

- Determining quantum length
  - Required time to process switch (context switch) : Balance between overhead and response time
    - e.g. 1ms for process switching:
      - 4ms quantum: overhead = 1 / (1 + 4) = 20%
      - 100ms quantum:
        - Overhead = 1 / (1+ 100) ≒ 1(%)
        - Maximum response (wait) time = ready queue length × 100 (ms)
  - Correlation with the mean CPU burst
    - If the quantum is set longer than the mean CPU burst, preemption will not happen very often.
      ⇒ most processes will perform a blocking operation before the quantum runs out, causing a process switch.
      ⇒ Phenomenon of Overhead

Quantum	Overhead	Response time
Short (Small)	Large (○)	Short (×)
Long (Large)	Small (×)	Long (○)

Priority Scheduling

- Round-robin scheduling had the assumption that all processes had equal priority. ⇒ Not appropriate in reality.
- Assign the priority for each process, and give priority CPU to the process with the higher priority.
- Manage a list (queue) of processes that can be executed for each priority.
- When the high-priority list is empty, process the low-priority list.
- Beware of “starvation” of low-priority processes.

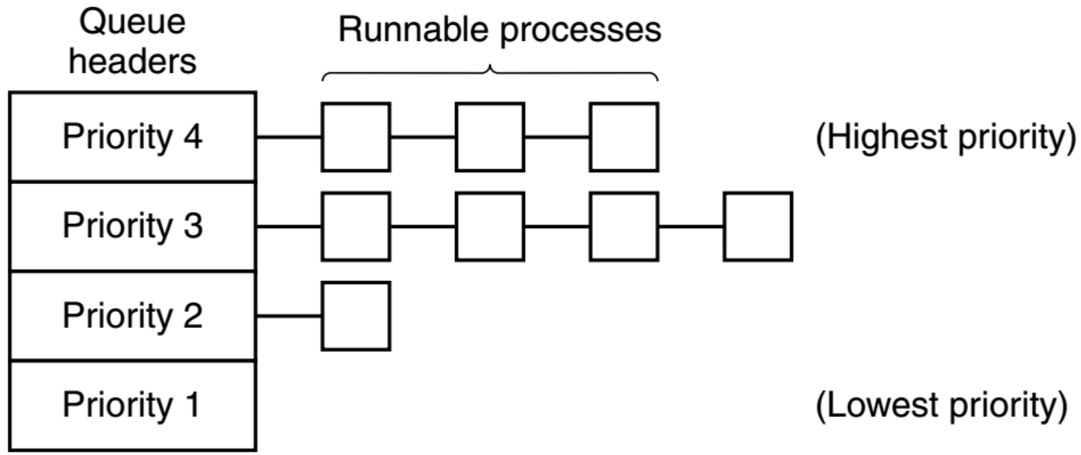


Figure 2-43. A scheduling algorithm with four priority classes.

Source: Andrew S. Tanenbaum. “Modern Operating Systems”, 4th ed., global ed., pp.161.

Example of Priority Scheduling: CTSS (classic)

- Setting priority classes (i.e. multiple queues)
- The priority of a process changes dynamically at runtime.
  - The priority of a process that has used up all of its CPU is lowered by one; the more I/O-bound a process is, the higher its priority becomes.
- Give long runtime to low-priority processes and short runtime to high-priority processes.
  - e.g. Priority#1 ⇒ 1 Quantum, Priority#2 ⇒ 2 Quantum, Priority#3 ⇒ 4 Quantum
- Reviving a low-priority process: Return to the highest priority when a new line is entered from the terminal !?

Example of Priority Scheduling: XDS940 (classic)

- 4 priority classes:
  - (High) Terminal (class)
  - I/O (class)
  - Short quantum (class)
  - (Low) Long quantum (class)
- Transition rules between classes:
  - A process that returns from waiting for terminal input goes to the "Terminal class”.
  - A process that returns from waiting for I/O to finish will go to the “I/O class”.
  - Processes that have used up all Quantum will go to the "Long Quantum class".

Shortest Process Next

- Even in interactive systems, the idea of prioritizing the shortest process can be used to improve the average response time.
- Definition of “shortest”: Treat each command execution as one job, and give priority to the command with the shortest execution time.
- Problem: The time required for a dynamically executed command is typically unknown.
- Example approach:
  - Estimate the execution time of a new command based on the execution time of commands in the past.
  - e.g. $T_n = \alpha T_{n-2} + (1 - \alpha) T_{n-1}$ α: 過去の履歴の参入度合

Guaranteed Scheduling

- Make some kind of agreement (contract) with the user and schedule the process so as to ensure the agreement is adhered to.
- Example agreement: 1/n (n: number of users currently logged in)
  - For each process, maintain the elapsed time from creation and the amount of running time.
  - Schedule to make this ratio 1/n.
- What is the actual scheduling algorithm?

Lottery Scheduling

- One of the possible methods of implementing guaranteed scheduling.
- Randomly select a process to execute. (Draw a “lottery.”)
  - e.g. 50 draws/sec ⇒ The winner gets 20ms of CPU time.
- Give additional “lotteries” for important processes.
  - e.g. From a lottery of 100 cards, give 20 cards to a process.
    ⇒ After a certain amount of time, the process will have utilized 20% of the total CPU time.

Fair-Share Scheduling

- Consider “fairness” in process selection.
- e.g. fairness = Same CPU utilization for each user
  - When a single user can run multiple processes concurrently, if the scheduling system focuses only on the processes, the CPU utilization of the user who runs many processes will increase.
  - Two users, A and B, each given 50% CPU time.
    A’s processes: A, B, C and D.　B’s process: E
    - A E B E C E D E A E B E C E D E A E …
  - When user A is given twice the CPU time.
    - A B E C D E A B E C D E A B E C D E …

Scheduling in Real-Time Systems

- Categories of real-time systems:
  - Hard real time
  - Soft real time
- Categories of processes on real-time system:
  - Periodic process
    - Processes that repeat some task periodically: e.g. Media play
  - Aperiodic process
    - Processes that handle asynchronous events (external events).
- “Schedulability” in real-time systems:
  - m periodic events and event i occurs with period P_i and requires C_i sec of CPU time to handle each event:

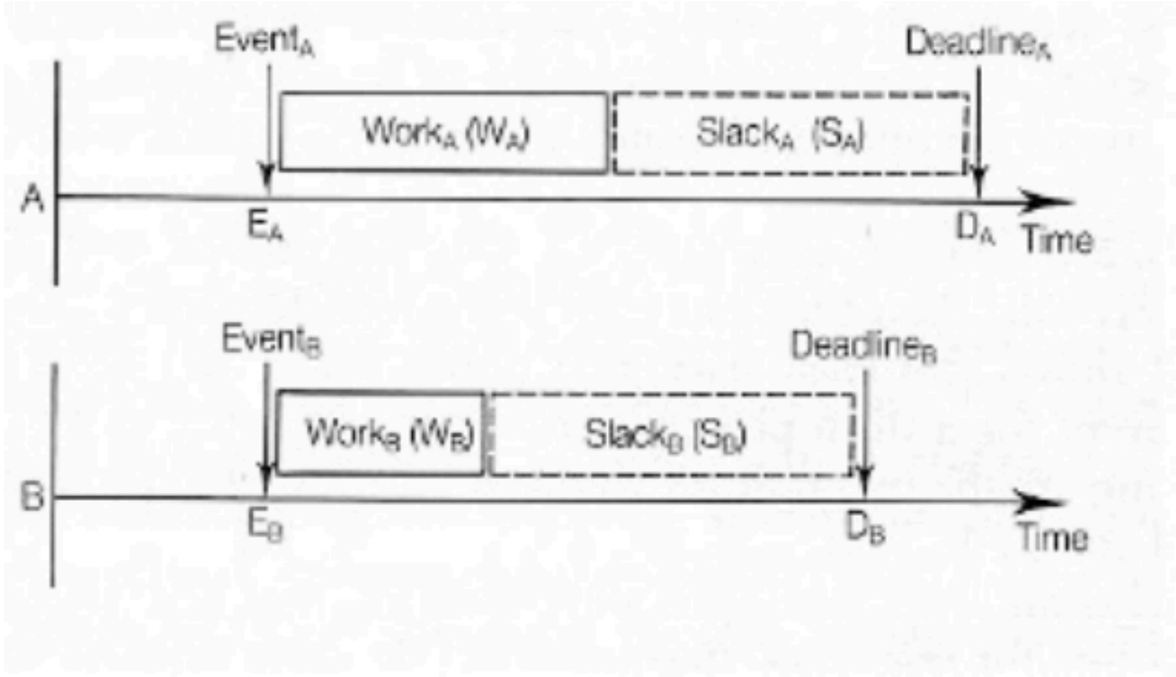
$$\sum_{i=1}^{m} \frac{C_i}{P_i} \leq 1$$

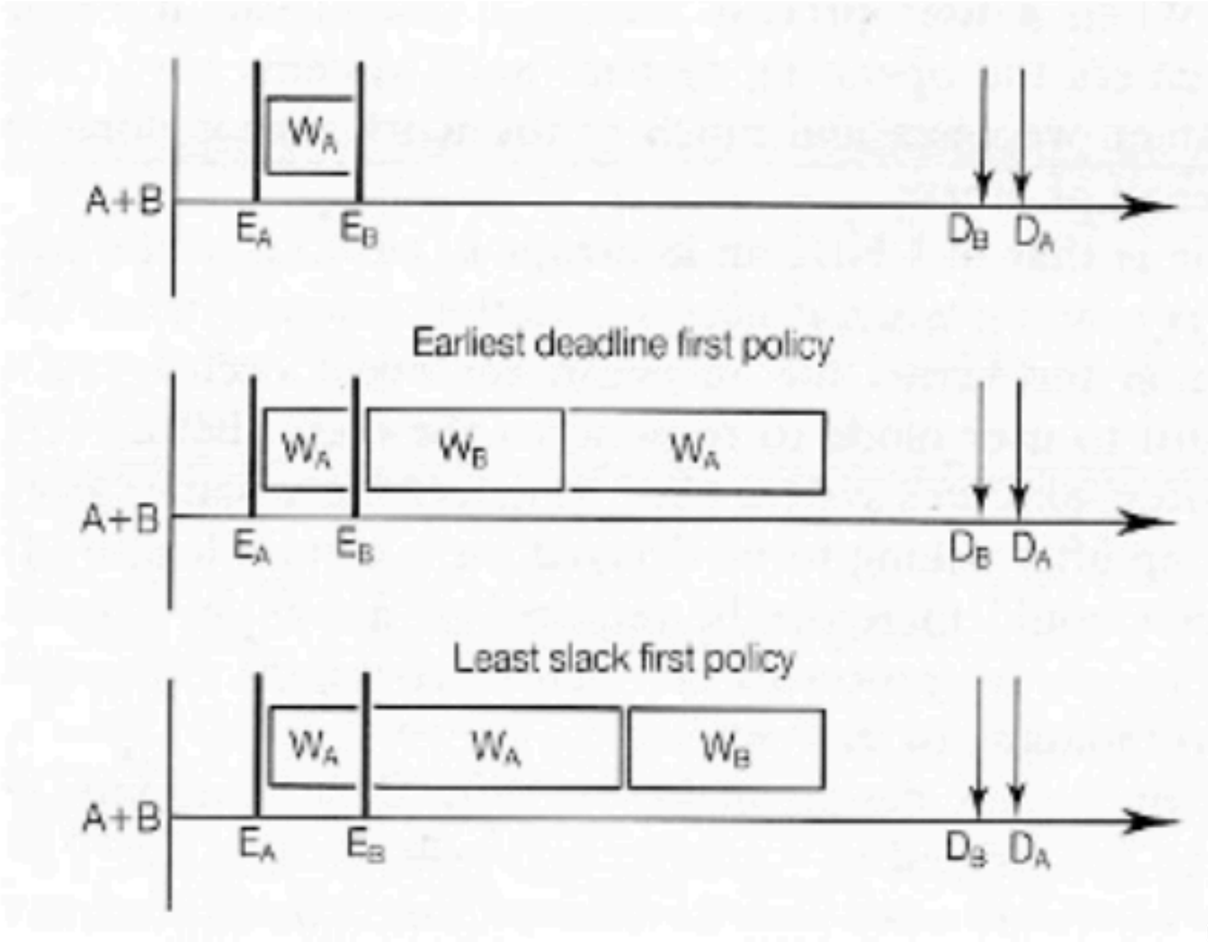
Example of real-time scheduling

- Choose based on the real-time requirements given to the system
- 100% utilization is possible:
  - Earliest Deadline First (EDF)
  - Least Slack Time (LST)
- CPU utilization is less than 100%:
  - Rate Monotonic Scheduling
  - Deadline Monotonic Scheduling

Example of real-time scheduling

- EDF v.s. LST





Separation of policy and mechanism

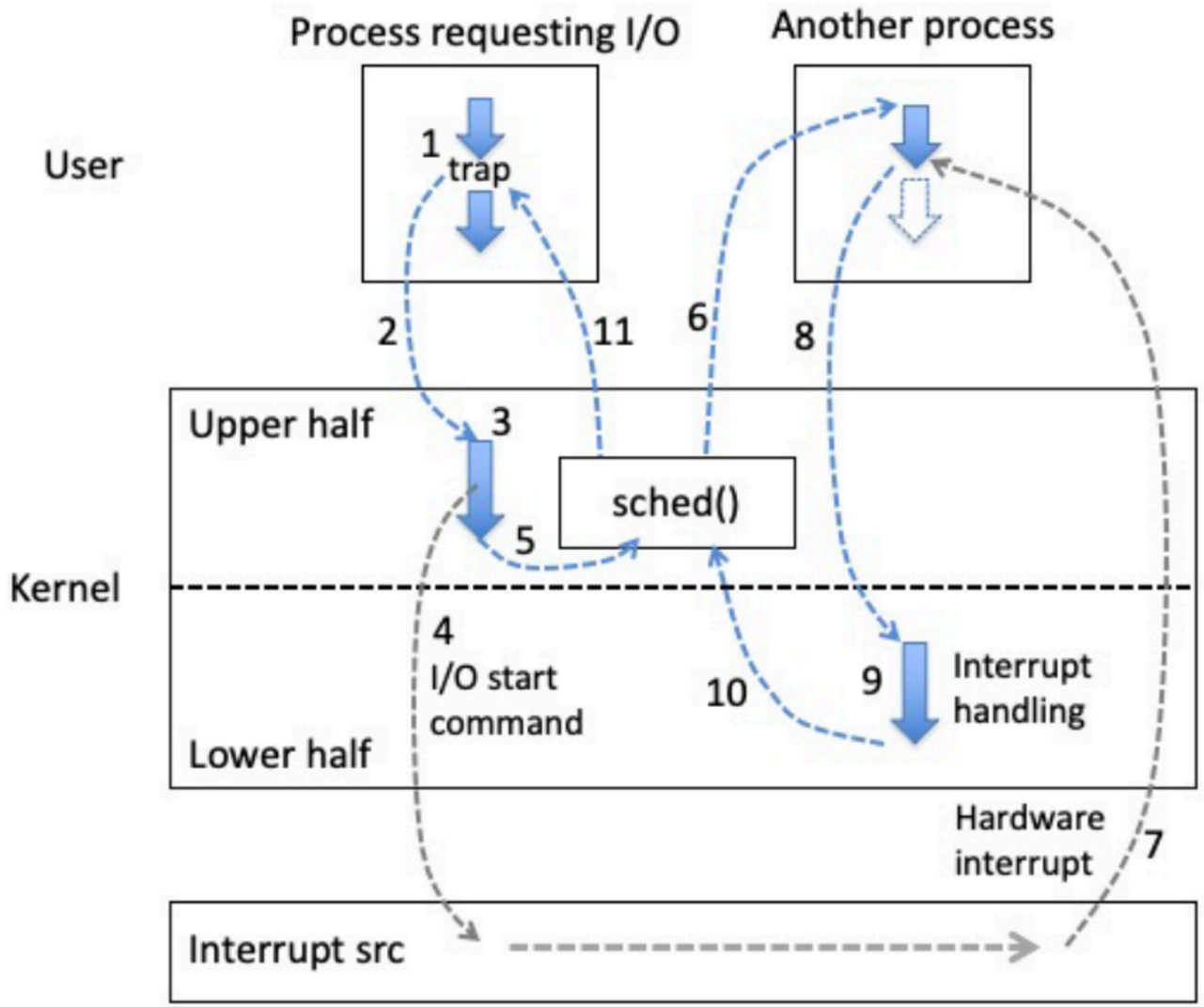
- Discussions on OS design methods
- Problem: Scheduling that is based on the behavior of processes has the potential to achieve a high level of objectives, but "observing the behavior" is not always easy.
- Solution: The OS provides only the scheduling mechanism, and the function to determine the actual scheduling parameters based on a certain policy is provided separately.
- e.g. Priority class on XDS940: Have the process declare to itself which class it is in.
- similar e.g. madvise on virtual memory.

日本語資料

2.4 スケジューリング

- 2.4.1 スケジューリングへの準備
  - プロセスの挙動
  - スケジュール時期
  - スケジューリングアルゴリズムの分類
  - スケジューリングアルゴリズムの目標
- 2.4.2 バッチシステムにおけるスケジューリング
  - 到着順サービス (First-Come, First-Served / First-In, First-Out)
  - 最短ジョブ優先 (Shortest Job First)
  - 最小残り時間優先 (Shortest Remaining Time Next)
- 2.4.3 会話型システムにおけるスケジューリング
  - ラウンドロビン (Round-Robin)
  - 優先度スケジューリング
  - 複数キュー
  - 最短プロセス優先 (Shortest Process Next)
  - 保証スケジューリング
  - 宝くじスケジューリング (Lottery Scheduling)
  - 公平な取り分スケジューリング (Fair-Share)
- 2.4.4 リアルタイムシステムにおけるスケジューリング
- 2.4.5 ポリシーとメカニズム
- 2.4.6 スレッドスケジューリング

スケジューラの位置と働き — I/O ブロックの例



背景

- CPU 時間は貴重なリソースである。
  - マイクロコンピュータの誕生は事情を少しだけ変えたが、それでもスケジューリングの重要性は変化していない。
- ユーザ体感の重視。
- プロセス切替(コンテキスト・スイッチ)に要する手間・時間。

プロセスの挙動

- 計算と入出力の繰り返し。
- CPU バウンドと I/O バウンド
  - CPU が高速になると I/O バウンドになりやすい。

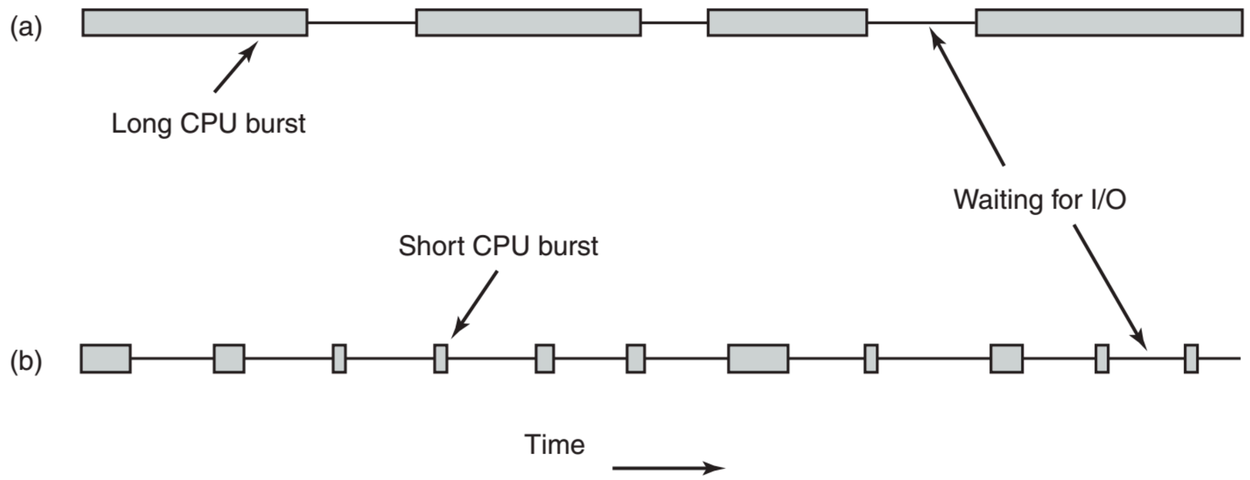


Figure 2-39. Bursts of CPU usage alternate with periods of waiting for I/O.
(a) A CPU-bound process. (b) An I/O-bound process.

Source: Andrew S. Tanenbaum. “Modern Operating Systems”, 4th ed., global ed., pp.151.

スケジュール時期 (タイミング)

1. 子プロセスが生成された場合。
2. 実行中のプロセスが終了した場合。
3. 実行中のプロセスが、入出力、同期機構、あるいは別の理由でブロックした場合。
4. 入出力操作が終了した場合。
5. クロック割り込みが発生した場合。

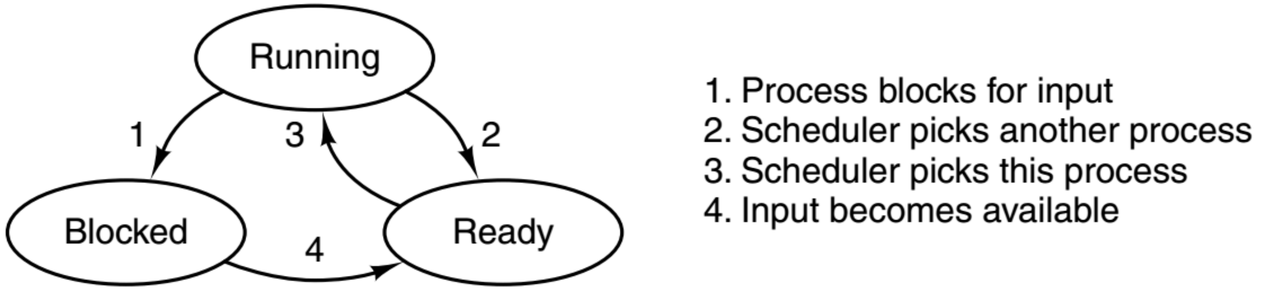


Figure 2-2. A process can be in running, blocked, or ready state. Transitions between these states are as shown.

Source: Andrew S. Tanenbaum. “Modern Operating Systems”, 4th ed., global ed., pp.93.

横取り v.s. 横取り無しスケジューリング

- 横取り無し (Non-preemptive)
  - 実行中のプロセスが自発的にブロックあるいは実行を放棄するまで、スケジューリングを行わない方式。
- 横取り (Preemptive)
  - 一回の実行を、高々ある決められた時間 (quantum — クァンタム) に限る方法。
  - クロック割り込みでスケジューリングが行われる。

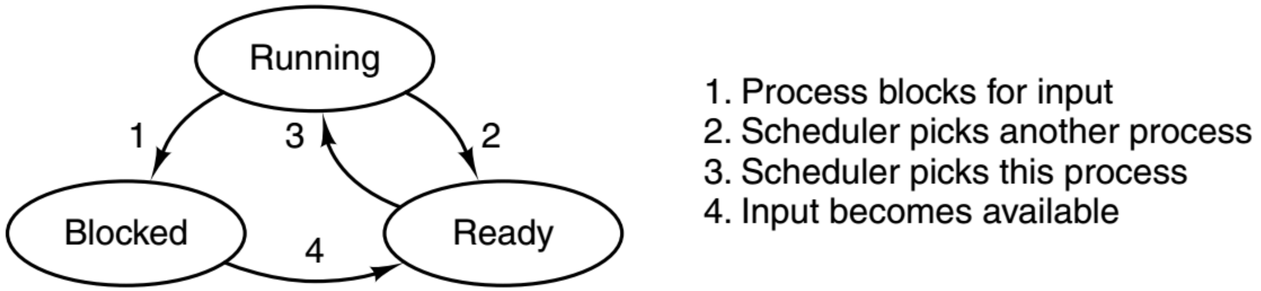


Figure 2-2. A process can be in running, blocked, or ready state. Transitions between these states are as shown.

Source: Andrew S. Tanenbaum. “Modern Operating Systems”, 4th ed., global ed., pp.93.

スケジューリングアルゴリズムの分類

- 環境が異なれば、異なるスケジューリングアルゴリズムが必要とされる。
- 3 種類の環境
  - バッチシステム
  - 会話型システム
  - リアルタイムシステム
- 複数のタイプの環境を同時に実現する必要のあるシステムも存在する。
- フェーズのあるプログラムの存在 (フェーズごとに異なる挙動を見せる場合がある)。

スケジューリングアルゴリズムでの考慮点

- スループット (Throughput)
  - 単位時間あたりに実行を完了したプロセス(ジョブ)の数。
- 遅延 (Latency — レーテンシ)
  - ターンアラウンド時間
    - プロセス(ジョブ)の投入から返却までの時間
  - 応答時間
    - 要求が起きてから最初の応答が得られるまでの時間
- 公平性 (Fairness)

スケジューリングアルゴリズムの目標 (1)

- 共通
  - 公平性
    - 各プロセスに公平に CPU を割り当てる。
  - ポリシーの実現
    - あらかじめ定められたポリシーが実現されているかどうか。
  - システム各部のバランス
    - システムの各部が常に最大限に動作中となるようにする。

スケジューリングアルゴリズムの目標 (2)

- バッチシステム
  - スループット — 単位時間当たりのジョブ実行数を最大化
  - ターンアラウンド時間 — 投入から返却までの時間を最小化
  - CPU 使用率 — CPU を常に最大限稼動
- 会話型システム
  - 応答時間 — 要求に対する応答を迅速に
  - 比例制 — ユーザの想定に従って
- リアルタイムシステム
  - デッドラインを守る — データ損失を防ぐ
  - 予測可能性 — マルチメディアシステムで品質低下を防ぐ

バッチシステムにおけるスケジューリング

- 代表的な 3 種類のバッチシステム用スケジューリングアルゴリズム
  - 到着順 (First-Come, First-Served)
  - 最短ジョブ優先 (Shortest Job First)
  - 最小残り時間優先 (Shortest Remaining Time Next)

到着順サービス (First-Come, First-Served)

- 横取り無し。
- CPU を要求した順番に CPU が割り当てられる。
- 実行可能プロセスは、単純なキュー構造。
- 長所
  - 単純である。
  - (ある意味) 公平である。
- 問題点
  - CPU バウンドプロセスと I/O バウンドプロセスに対する扱いの差。
  - 例) 1 秒間実行した後、1 回だけディスク I/O を行うことを繰り返すプロセス A と、ほとんど CPU を使用せず I/O を 1000 回繰り返すプロセス B。
    - B は 1 秒おきに 1 回の I/O しか実行できない (終了までに 1000 秒)。
    - もし 10 ミリ秒ごとに A から CPU を横取りして B に与えると、B は 10 ミリ秒ごとに 1 回の I/O が実行でき 10 秒で終了できる。
    - この場合でも、A には即座に CPU が渡されるため、A の終了までの時間は、ほとんど変化が無い。

最短ジョブ優先 (Shortest Job First)

- あらかじめ所要時間がわかっていると仮定。
- 横取り無し。
- 所要時間が短いものから CPU を割り当てる。
- 特徴
  - 平均的な待ち時間 (ターンアラウンド時間) を短縮することができる。

Figure 2-41. An example of shortest-job-first scheduling. (a) Running four jobs in the original order. (b) Running them in shortest job first order.

Source: Andrew S. Tanenbaum. “Modern Operating Systems”, 4th ed., global ed., pp.157.

最小残り時間優先 (Shortest Remaining Time Next)

- 最短ジョブ優先の横取り版。
- 新たに到着したジョブを、最短ジョブ優先のアルゴリズムに組み込むことができる。

会話型システムにおけるスケジューリング

- ラウンドロビン (Round-Robin)
- 優先度別
  - 複数のキュー
- 最短プロセス優先
- 保証 (契約)
- 宝くじ (確率的)
- 公平分配

ラウンドロビン (Round-Robin)

- 実行可能プロセスは線形リスト(キュー)で管理される。
- プロセスには、クォンタム(quantum)と呼ばれる、ある一定の時間単位で CPU が与えられる。
- クォンタムを使い切ったプロセスからは CPU が横取りされ、プロセスはキューの最後につながれ、キューの先頭から次のプロセスが取り出されて CPU が与えられる。

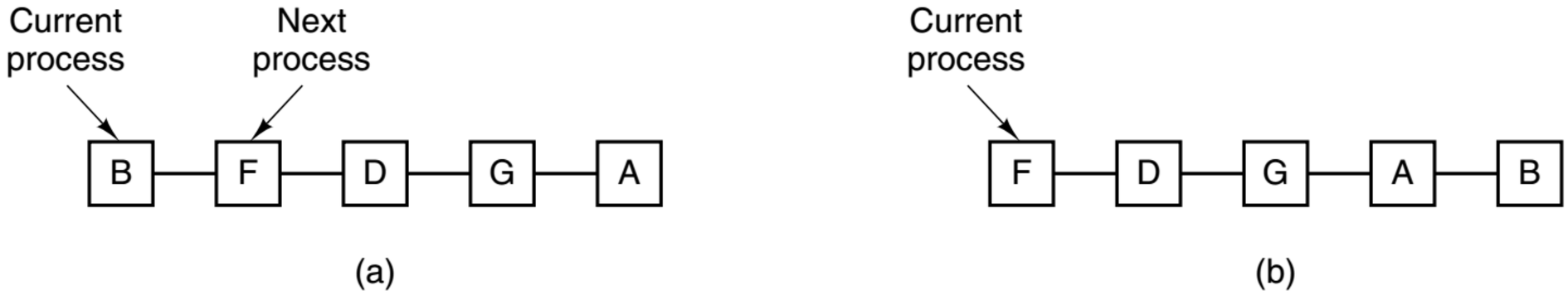


Figure 2-42. Round-robin scheduling. (a) The list of runnable processes. (b) The list of runnable processes after *B* uses up its quantum.

Source: Andrew S. Tanenbaum. “Modern Operating Systems”, 4th ed., global ed., pp.159.

ラウンドロビンに対する疑問

- クォンタムの値はどのように決定するのか?
- 全てのプロセスは等しく扱われるべきか?
- I/O 完了に伴い実行可能となったプロセスは、リストのどの位置(先頭? 最後尾?) に置くべきか?

ラウンドロビンの解析

- クォンタムの設定
  - プロセスの切り替え (process switch = コンテキスト切り替え: context switch) に要する時間 = オーバーヘッドと応答性能の兼ね合い。
    - 例: プロセス切り替えに 1ms を要する場合
      - クォンタム 4ms: オーバーヘッド = 1 / (1 + 4) = 20%
      - クォンタム 100ms:
        - オーバーヘッド = 1 / (1+ 100) ≒ 1(%)
        - 最大待ち時間 = 実行可能リストの長さ × 100 (ms)
  - 平均的な CPU バーストとの関係
    - 平均的な CPU バーストよりも長く設定された場合、CPU の横取りは生じにくくなる
      ⇒ プロセスがブロックした場合にプロセス切り替えが起きる可能性が高くなる
      ⇒ オーバーヘッドの現象

Quantum	Overhead	Response time
Short (Small)	Large (○)	Short (×)
Long (Large)	Small (×)	Long (○)

優先度スケジューリング (Priority Scheduling)

- ラウンドロビンには、全てのプロセスの優先度が等しいという仮定があった: 現実には適当ではない。
- プロセスごとに優先度を割り当て、優先度の高いプロセスに優先的に CPU を与える。
- 優先度ごとに実行可能なプロセスのリスト(キュー)を設ける。
- 優先度の高いリストが空の場合に、優先度の低いリストを処理する。
- 低優先度プロセスの「飢餓状態」に注意。

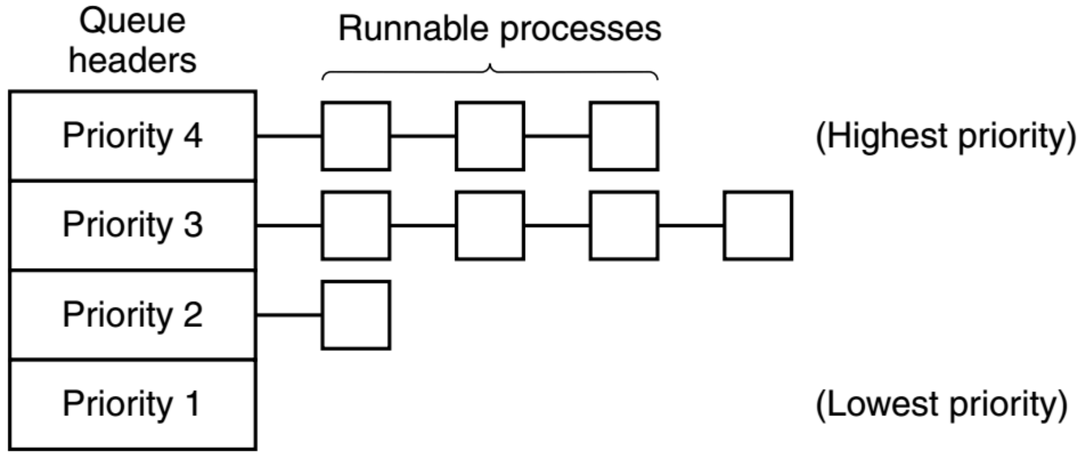


Figure 2-43. A scheduling algorithm with four priority classes.

Source: Andrew S. Tanenbaum. “Modern Operating Systems”, 4th ed., global ed., pp.161.

優先度スケジューリングの例: CTSS (古典)

- 優先度クラス(複数キューに対応)の設定。
- プロセスの優先度は実行時に動的に変化する。
  - CPU を使い切ったプロセスの優先度は 1 つ下がる: I/O バウンドなプロセスほど高い優先度に設定される。
- 低優先度のプロセスには長い実行可能時間、高優先度のプロセスには短い実行可能時間を与える。
  - e.g. 優先度#1 ⇒ 1 クォンタム, 優先度#2 ⇒ 2 クォンタム, 優先度#3 ⇒ 4 クォンタム
- 低優先度になったプロセスを復活させる方法: 端末から改行が入力された際に最高優先度に戻す!?

優先度スケジューリングの例: XDS940 (古典)

- ４ 種類の優先度クラス:
  - (高)　端末 (クラス)
  - 　　I/O (クラス)
  - 　　短クォンタム (クラス)
  - (低)　長クォンタム (クラス)
- クラスの遷移ルール
  - 端末の入力待ちから戻ったプロセスは端末クラスへ
  - I/O の終了待ちから戻ったプロセスは I/O クラスへ
  - クォンタムを使い切ったプロセスは長クォンタムクラスへ

最短プロセス優先 (Shortest Process Next)

- 会話型システムでも、最短プロセスを優先するという考え方は、平均的な応答時間を改善するために利用できる。
- 最短の定義: 1 回のコマンド実行を 1 つのジョブと見なし、実行時間が最短のコマンドを優先する。
- 問題: 動的に実行されるコマンドの所要時間は、一般的には不明である。
- 解の例:
  - 過去に実行されたコマンドの実行時間を参考に、新しいコマンドの実行時間を予測する。
  - 例) $T_n = \alpha T_{n-2} + (1-\alpha) T_{n-1}$　　α: 過去の履歴の参入度合

保証スケジューリング (Guaranteed Scheduling)

- ユーザとの間で何らかの取り決め(契約)を交わし、その取り決めが守られるようにスケジューリングを行う。
- 契約の例: 1/n (n: 同時にログインしているユーザ数)
  - プロセスごとに、生成されてからの時間と実行できた時間を管理する。
  - この比が 1/n になるようにスケジューリングを行う。
- 実際のスケジューリング方法は?

宝くじスケジューリング (Lottery Scheduling)

- 保証スケジューリングの実現方法のひとつ。
- 実行するプロセスを、ランダムに選択する (「くじ」を引く)。
  - 例: 毎秒 50 回のくじ引き、勝者に 20 ms の CPU 時間が与えられる。
- 重要なプロセスには、追加の「くじ」を与える。
  - 例: 100 枚のくじから、あるプロセスに 20 枚を与える
    ⇒ 一定の時間の後では、そのプロセスは全体の 20% の CPU を使用している。

公平共有 (Fair-Share)

- プロセスの選択に、「公平性」を考慮。
- 例: 公平性 = ユーザごとの CPU 利用率が同じ。
  - 1 人のユーザが複数のプロセスを同時並行実行することが可能なとき、プロセスのみに注目してスケジューリングすると、プロセスを多く走らせたユーザの CPU 利用率は上昇する。
  - 2 ユーザ A と B, CPU はそれぞり 50% ずつ。
    A のプロセス: A, B, C, D.　B のプロセス: E
    - A E B E C E D E A E B E C E D E A E …
  - 上でユーザ A に 2 倍の CPU を与えたい場合
    - A B E C D E A B E C D E A B E C D E …

リアルタイムシステムにおけるスケジューリング

- リアルタイムシステムの分類
  - ハードリアルタイムシステム
  - ソフトリアルタイムシステム
- リアルタイムシステムにおけるプロセスの分類
  - 周期プロセス (Periodic process)
    - 何らかの処理を周期的に繰り返すプロセス: e.g. メディア再生
  - 非同期プロセス (Aperiodic process)
    - 非同期イベント(外部イベント)の処理を行うプロセス
- リアルタイムシステムにおけるスケジューリング可能性
  - あるモデル: m 個の周期プロセス、周期 P_i 、実行時間が C_i であった場合:

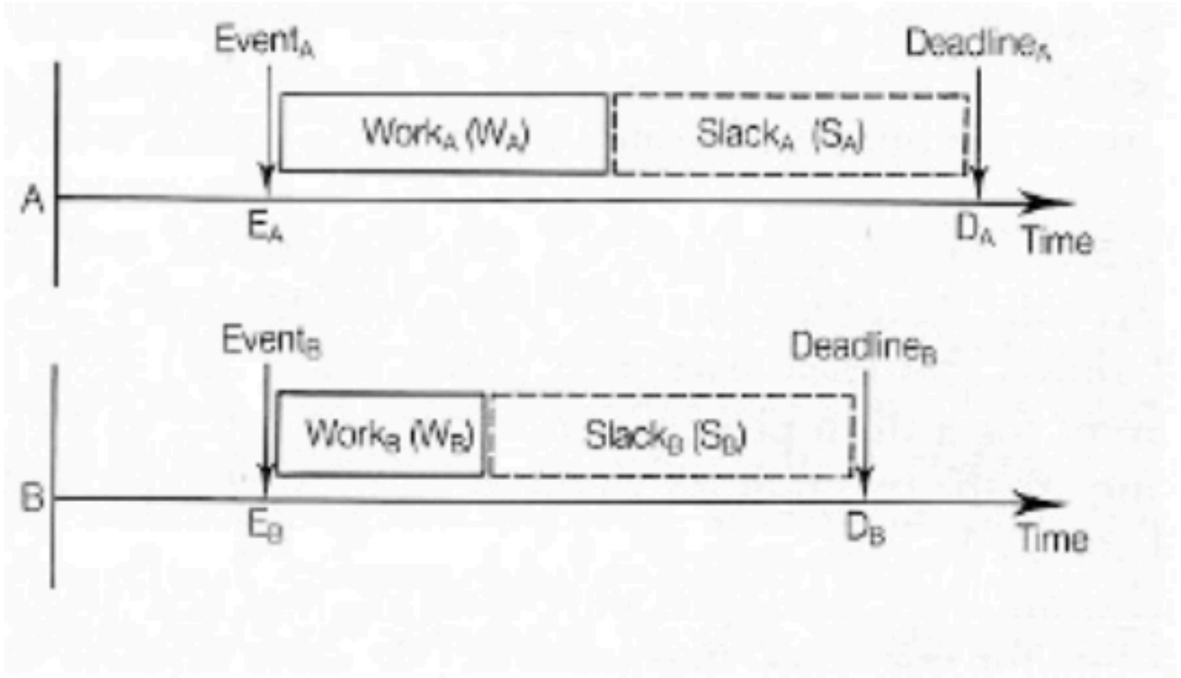
$$\sum_{i=1}^{m} \frac{C_i}{P_i} \leq 1$$

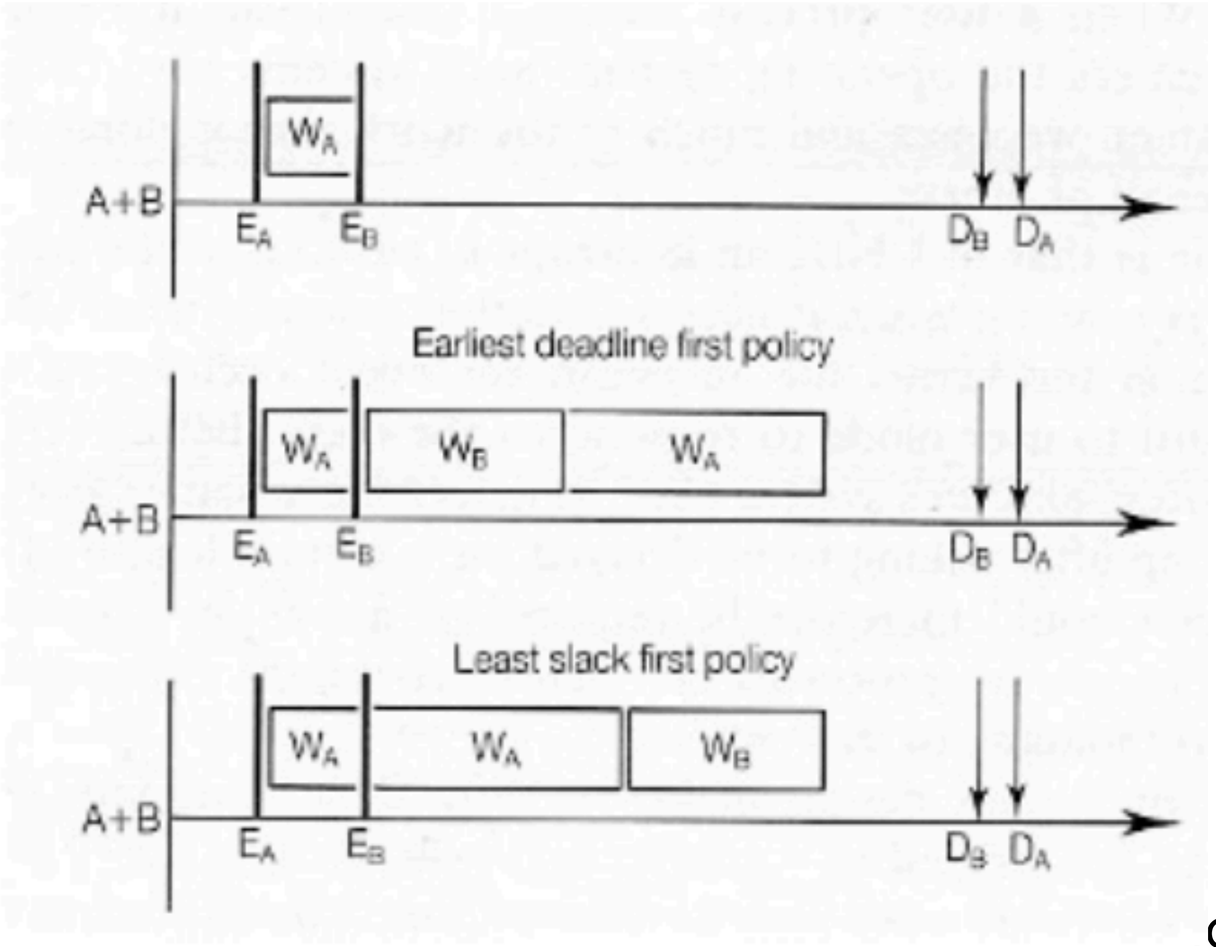
リアルタイムスケジューリングの例

- システムに与えられるリアルタイム要求に基づいて選択
- 100% 利用率が可能なもの:
  - 最近デッドライン優先 (Earliest Deadline First — EDF)
  - 最小余裕優先 (Least Slack Time — LST)
- CPU 利用率が 100% 以下のもの:
  - レート単調 (Rate Monotonic Scheduling)
  - デッドライン単調 (Deadline Monotonic Scheduling)

リアルタイムスケジューリングの例

- EDF v.s. LST





ポリシーとメカニズムの分離

- OS の設計方法に係わる議論。
- 問題: プロセスの挙動に基づいて実行するスケジューリングは、高いレベルで目的を達成できる可能性があるが、「挙動を観察する」ことは必ずしも容易ではない。
- 解: OS では、スケジューリングのメカニズムだけを提供し、あるポリシーに基づいて実際のスケジューリングパラメータを決定する機能は別に設ける。
- 例: XDS940 タイプの優先度クラス: どのクラスにいるのかを、プロセス自身に申告させる。
- 同様な例: 仮想メモリにおける madvise