

KYUSHU UNIVERSITY · SYS-104

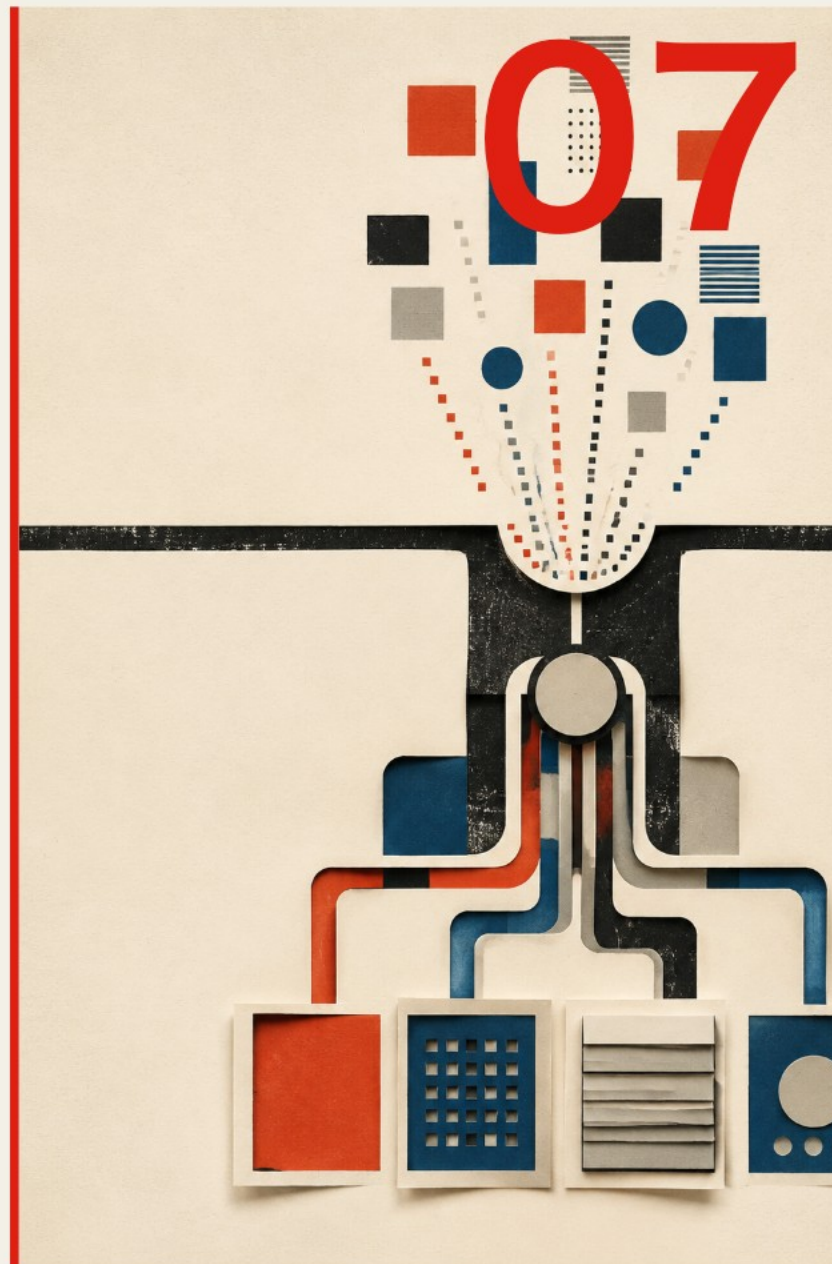
オペレーティング システム

OS の役割と仕組み

2026 年 9 月

Olivia Chen Toyama · C2LAB

LECTURE 7



マンションの管理に例えると



暮らしは、それぞれ

仕事、読書、洗濯
目的もペースも違う

設備は、みんなで

エレベーターや共用部分を
同じ建物で共有する

共用部分には、管理が必要

利用のルール、設備の保守
困ったときの相談窓口

アプリも、**独立して動きながら、同じハードウェアを使う**

同じパソコンで、三つの仕事



ブラウザ

クリックには、すぐ反応してほしい

画像編集

大きな写真の計算を進めたい

ファイル同期

保存や通信の完了を待つ

OS = Operating System (オペレーティングシステム)

複数のアプリがハードウェアを使えるように管理する基本ソフトウェア

保存するとき、OSが間に入る



パソコン

Windows・macOS・Ubuntuなど

スマートフォン

Android・iOS

 **アプリ**
文書を保存したい

↓ 保存を頼む

 **OS**
お願いを受けて、機器を動かす

↓ 機器を制御する

 **ハードウェア**
SSD にデータを残す

アプリと機器の間で、**OS が処理を引き受ける。**

講義の流れ

身近な操作から、OS の働きを見つける。

01 OSの役割

なぜ必要？

02 CPU

どう交代する？

03 メモリ

どう分ける？

04 ファイル

どう保存する？

05 操作

どう頼む？

06 歴史

なぜこの形？

OSの基本的な役割

一台のパソコンを、複数のアプリで使っている。

アプリのお願いを、誰が引き受ける？



アプリ

OS がつなぐ



CPU・メモリ・機器

01 **OSの役割**

02 CPU

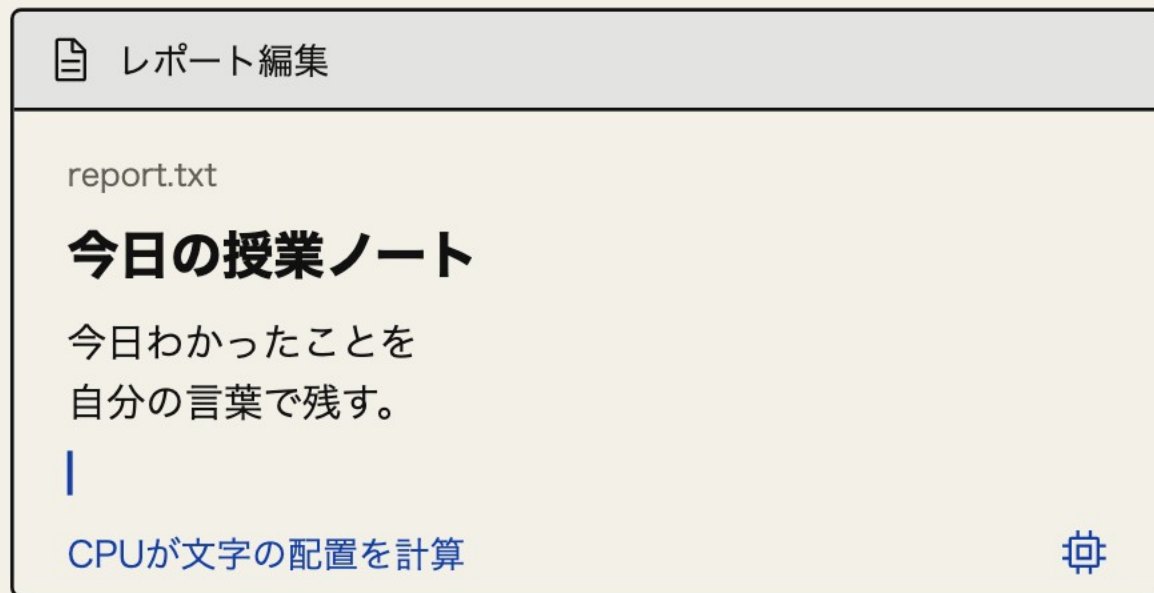
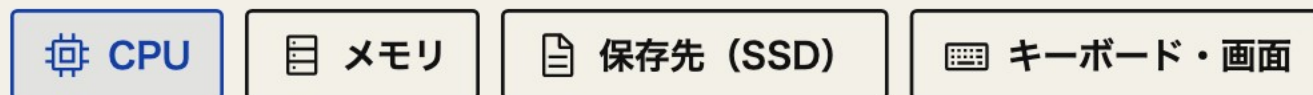
03 メモリ

04 ファイル

05 操作

06 歴史

レポートを書くにも、資源が必要



命令を実行する

文字を並べ、画面を更新する計算を実行

一つのアプリでも、**計算・作業場所・保存・入出力**が必要

CPUは交代、メモリは分けて使う

**ブラウザ**

CPUを使用中

**画像編集**

実行を待つ

**ファイル同期**

通信待ち

CPUは、短い時間ごとに交代

1 表示	2 編集	3 表示	4 編集	5 同期	6 表示
---------	---------	---------	---------	---------	---------

1コアの簡略例 / マスは時間の区切り、実測値ではない

交代しても、作業データは残す

ブラウザの領域	画像編集の領域	ファイル同期の領域
---------	---------	-----------

メモリは領域を分けて保持する

時間は交代、作業場所は分離

OSが両方を管理する

最初

戻る

再生

1/6

次へ

機器が違っても、同じ操作で開く



→
同じ「開く」
OSが機器ごとの
手順に振り分ける
→



内蔵SSD

内蔵ストレージ用の制御

読み出し要求を待つ

アプリは機器ごとの細かい手順を持たず、**共通の操作を使える**

別のアプリの作業を、勝手に壊させない



ブラウザ

開いているページ



勝手な書き換えを止める



写真編集

編集中的の写真

OS の中で、管理を担う中核

カーネル

使える場所や、できる操作を管理する。

CPU の保護機能と OSが、別々の作業を守る。

アプリは、OSに読み込みを頼む

✓ この文書を読む許可がある

report.txt を開いて読む



まずは、お願いがどこへ届くかを見よう。

文書アプリが、OS に読み込みを頼む。

最初

戻る

再生

1/3

次へ

機器からの知らせで、OSが動く

時間が来た

機器から返事が来た



機器からの知らせ

CPU の交代時刻



OS が動く

次の作業を選ぶ



その結果

ブラウザにも番が回る

このように機器が CPU に処理を知らせる仕組みを、**割り込み**と呼ぶ。

アプリのお願いだけでなく、**機器からの知らせ**でも OS が動く。

OSの三つの役割



資源の配分

分ける

表示

編集

表示

CPUの時間は交代
メモリは領域を分けて保持



共通の操作

そろえる

同じ「開く」 → **SSD / USB**

機器が違ってても
アプリは共通の操作を使う



保護

守る

不正な操作 × **停止**

他のデータへの書き込みや
勝手な機器操作を防ぐ

次は「分ける」を具体化：**複数のプログラムで、CPUをどう使うか**

CPUを使う順番を決める

OS は、いくつかのアプリからお願いを受け取る。

写真の処理中も、ブラウザを使いたい。



写真を計算する

CPU をどう分ける？



クリックに反応する

01 OSの役割

02 **CPU**

03 メモリ

04 ファイル

05 操作

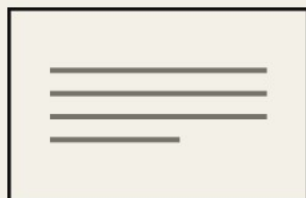
06 歴史

起動すると、作業場所ができる



SSD に保存

写真編集のプログラム



まだ、作業は始まっていない。

起動する →

起動すると…



ここで、作業が始まる。

保存されたプログラムを起動すると、**作業中の状態が生まれる。**

OSは、作業のデータと続きを管理する

写真編集のプロセス

OS が、ひとまとまりとして管理する



すること

写真を読む

明るさを変える

画面に表示する



作業中のデータ



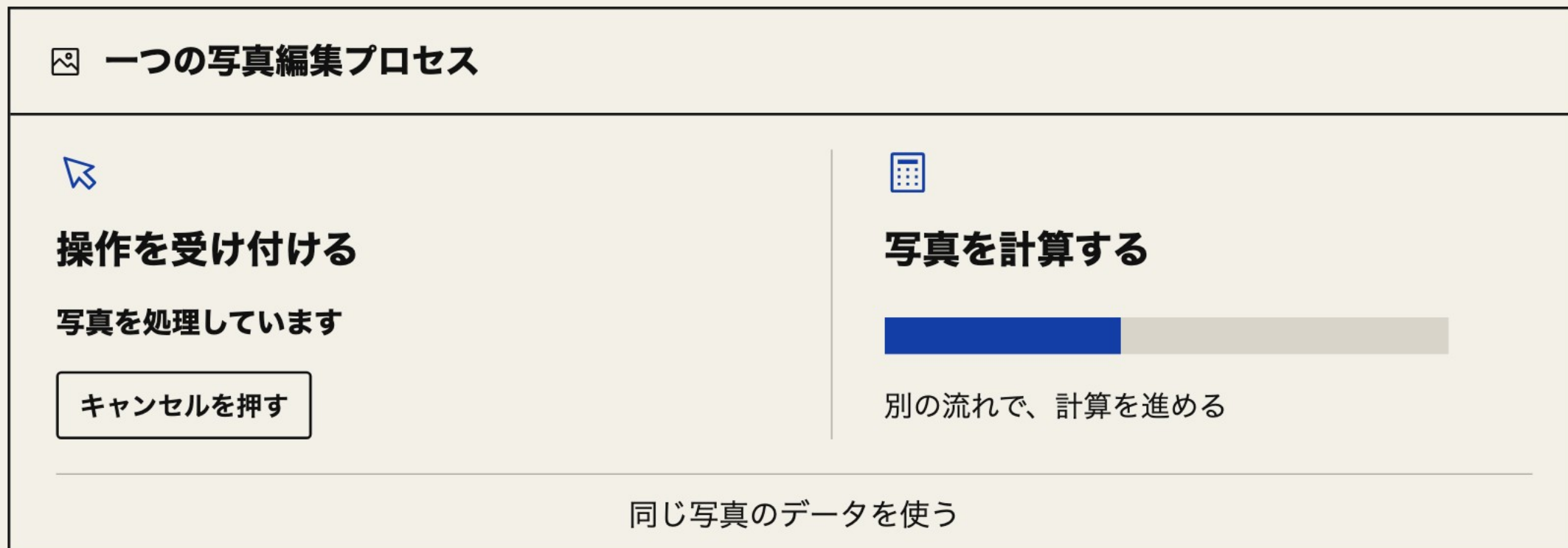
どこまで進んだか

次にすること

画面に表示する

OS は、**データと作業の続き**を、アプリごとに管理する。

写真の処理中も、操作を受け付けたい



一つのプロセスの中の実行の流れを、**スレッド**と呼ぶ。

交代して進む。同時にも進める。

CPU の処理担当（コア）が 1 つ

交代しながら進む



時間 →

並行

CPU の処理担当（コア）が 2 つ

同じ時刻に進む

コア 1

写真

コア 2

ブラウザ

並列

「複数の作業が進んでいる」にも、**進め方の違いがある。**

「待つ」にも、二つの理由がある



ブラウザの今

CPU の順番待ち

実行可能

CPU を使っているのは

写真編集



SSD

順番待ち

計算する

返事待ち

順番待ち

再開

写真編集が CPU を使っている。ブラウザは順番を待つ。

最初

戻る

再生

1/5

次へ

同じ仕事でも、順番で待ち時間が変わる

到着順・FCFS

順番に交代・RR

最後まで見る



時刻 0/12



CPU の今の担当

画像処理

先に到着した仕事を、最後まで実行する

0 1 2 3 4 5 6 7 8 9 10 11 12

画像処理

0 に到着・7 目盛り分

ブラウザ

1 に到着・2 目盛り分

同期の準備

2 に到着・3 目盛り分

👁 ブラウザが、最初に反応するまで

まだ待っている

同じ仕事でも、CPU を渡す順番で、使い心地が変わる。

1 コアで比べる説明用の時間。実際の秒数ではない。

交代するときは、続きの位置を保存

写真編集

1. 写真を読む
2. 明るさを計算
3. 計算結果を残す
4. 画面に表示する



交代するときに残す記録

次は 4 番目

計算途中の値も残す



CPU の今の担当

写真編集

写真の処理を進める

この切り替えが **コンテキストスイッチ**

写真の処理を、3 番目の手順まで進めた。

最初

戻る

再生

1/3

次へ

どちらなら、クリックに早く反応する？

到着順



順番に交代



比べるのは、**ブラウザに最初の番が回ってくるまで。**

[CPU の交代を自分で試す →](#)

作業中のデータを、分けて置く

CPU は交代。作業データは、メモリ（RAM）に残しておく。

別々の作業のデータは、混ざらない？



ブラウザのデータ

作業場所をどう分ける？



写真編集中的数据

01 OSの役割

02 CPU

03 **メモリ**

04 ファイル

05 操作

06 歴史

自分の番号と、実際の置き場所は別



利用者が覚えるのは

自分の番号

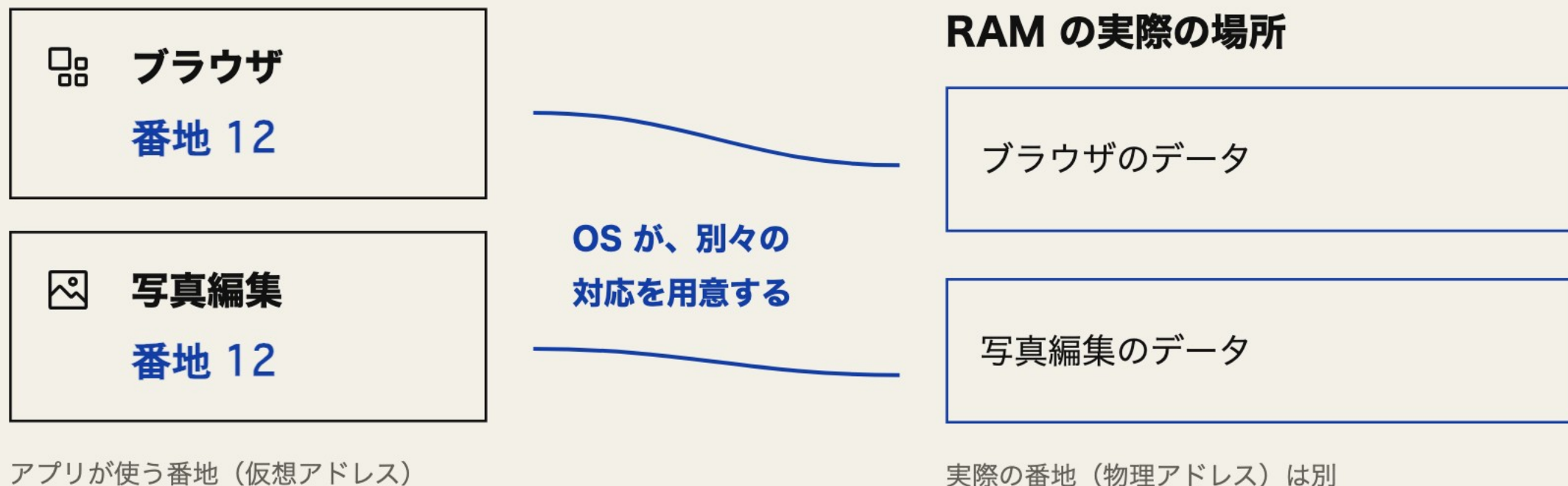
↓ 受付が、置き場所を探す

荷物が置いてあるのは

実際のロッカー

アプリも、**自分が使う番地**でデータを頼める。

同じ番地でも、別の作業場所につながる



同じ「番地 12」でも、**別々の作業場所**につながる。

区画の番号を変え、中の位置はそのまま

アプリが使う番地

ページ **1**



区画の先頭から **12** の位置

OSが管理する対応表

1 → **5**

置き場所を
対応させる

RAMでの置き場所

RAM の区画 **5**



区画の先頭から **12** の位置

同じ大きさの区画どうしを、対応させる。

区画の番号を対応させても、中の位置は変わらない。

区画の先頭部分だけを示した模式図。

番地から、RAM の置き場所をたどる

使うアプリ

写真編集 ▾

同じ「ページ 1」でも、アプリを変えると？

このアプリの対応表

アプリが頼む場所

ページ 1



ページ	RAM の区画
0	1
1	5
2	6



RAM の置き場所

区画 0	区画 1
区画 2	区画 3
区画 4	区画 5
区画 6	区画 7

その中の位置 12

写真編集が、ページ 1 の中の「位置 12」を使いたい。

最初

戻る

再生

1/3

次へ

RAM がないときは、どうする？

まだ RAM がない

権限がない



STEP 0/4



アプリ

必要なデータを使う

実行

読み込んでよいデータ

OS が引き受ける

必要なデータを用意

データの場所： まだ RAM がない

ページを用意する

RAM の一部

空き

使用中

使用中

使用中

SSD 必要なデータ



アプリが、必要なデータを使おうとする。

保存済みのデータを読み込む例。

RAM の外へ退避することもある



RAM

いま使うデータの場所

写真のデータ
ブラウザのデータ
今は使わないデータ



退避できる

データ移動には
時間がかかる



SSD など

一時的な退避先



退避用の場所

スワップ

RAM がいっぱい。別の作業の場所が必要になった。

最初

戻る

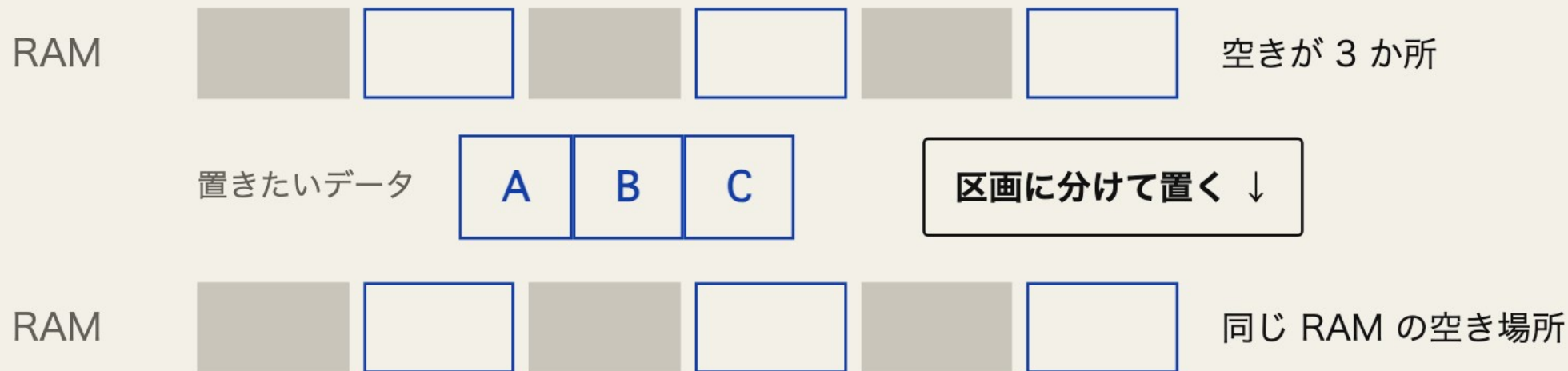
再生

1/3

次へ

空きがばらばらでも、分けて置ける

空きはある。でも、ひと続きではない。



データを同じ大きさの区画に分けると、**離れた空き場所にも置ける。**

一つを閉じても、別の作業は続く



ブラウザ

開いているページ

ブラウザを閉じる →



写真編集

編集中的写真

写真の作業まで、消えてしまう？

次は、作業中のデータを**ファイルとして残す仕組み**を見る。

ファイルと入出力の管理

ここまでは、メモリにある作業中のデータを見た。

保存した文書を、もう一度開くには？



文書の名前

名前であとる



保存されたデータ

01 OSの役割

02 CPU

03 メモリ

04 **ファイル**

05 操作

06 歴史

report.txt を開いて読む



名前で探す

report.txt という名前から、保存されたデータを指定

共通の操作で扱う

開く、読む、書く、閉じる

内部の配置は任せる

SSD のどの区画にあるか、アプリが直接管理しなくてよい

ファイル＝保存データを、名前や共通の操作で扱う仕組み

同じ名前のファイル、どう見分ける？



同じ名前でも、別のファイル。

ファイルをまとめる入れ物

ふだん使うフォルダのこと。

OS ではディレクトリと呼ぶ。

授業用の文書を開きたい

report.txt だけでは、どちら？

どのフォルダの中かも伝えよう。

ファイル名だけでなく、置いてある場所も必要。

[道順を文字で見る →](#)

ファイルには中身以外の情報もある

中身

report.txt

第7回の講義メモ…

種類	通常のファイル
大きさ	18 KB
持ち主	olivia
アクセス権	持ち主は読み書きできる
更新時刻	10:24

開いたファイルには、受付番号が付く



open("report.txt")

OS がファイルを探し、権限を確認して開く

成功すると、たとえば 3 が戻る

このプロセスの「開いているファイルの表」で、3 がそのファイルを指す

read(3) / close(3)

以後は、その番号を使って読む・閉じる

ファイルディスクリプタ (fd) = 開いたファイルなどを指定する、小さな整数

一回の読み込みを、最後まで追う

SSD から読む

RAM にある場合



STEP 0/6



アプリ側

文書アプリ

開いてある文書を読む



OS側

OS

文書を確認、読み込みを手配



機器

SSD

保存したデータを読む

要求 → SSD の処理 → 完了通知

文書アプリの状態

実行中 (RUNNING)

開いてある文書を、受け取った番号 3 で指定する

保存方式や機器の違いは OS が引き受け、アプリヘデータを渡す

その間の CPU

命令を実行する

誰が、何をできるか



アクセス権だけでは守れない



削除後に何が起こる？

NAMESPACE

名前との対応を外す

データ本体の扱いは **file system** による

TRASH

別の場所へ移す

アプリや **desktop** の機能

STORAGE

SSD・暗号化・上書き

復元可能性は媒体と設定で変わる

人が操作するための窓口

ファイルを開く・読む・保存する。その操作を、人から伝える。

コンピュータに、どうやって指示を伝える？



画面の対象を選ぶ

人とコンピュータの窓口



文字で操作を指示する

01 OSの役割

02 CPU

03 メモリ

04 ファイル

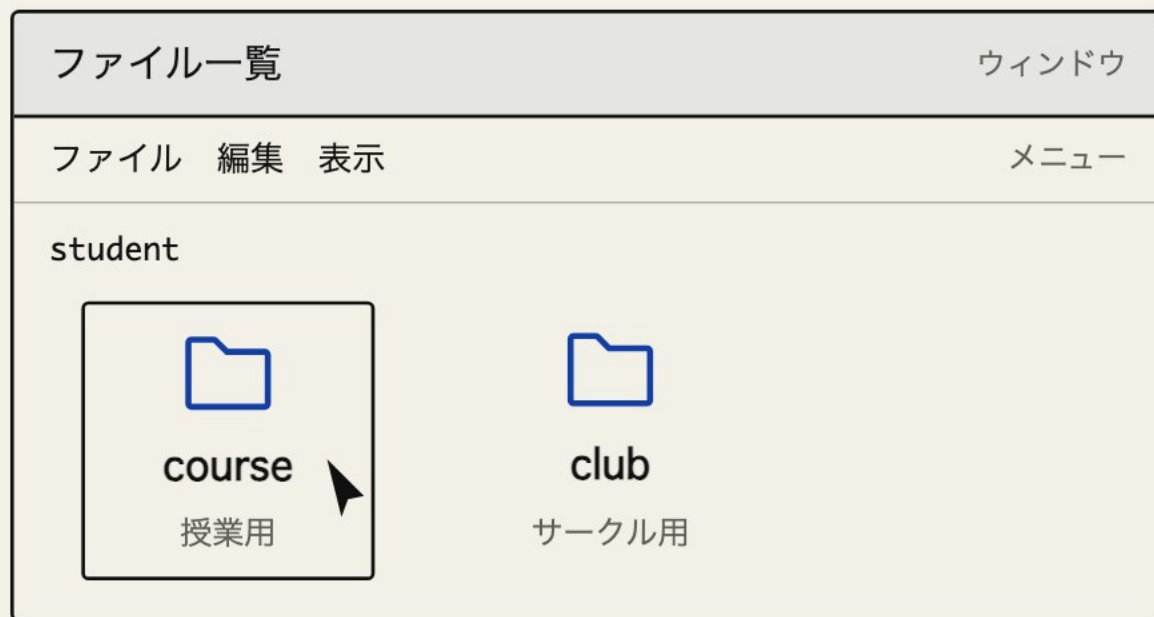
05 **操作**

06 歴史

GUI：画面の対象を選んで操作する

GUI Graphical User Interface

画面のアイコンやメニューを選んで操作する方式



選べるものが見えるので、**操作の手がかりを探しやすい。**

ふだんのフォルダ操作も GUI

course のアイコンを選んで開こう。

ウィンドウ	作業する画面の枠
アイコン	ファイルや機能を表す絵
メニュー	選べる操作の一覧
ポインタ	画面上の対象を指す印

CLI：文字で指示を入力する

CLI Command Line Interface

操作を表す文字列（コマンド）を入力する方式

端末 / student の中	授業用の模擬端末
<pre>\$ ls</pre>	実行 (Enter)
ls と入力して、Enter を押す。	

まずは、一つだけ試す

ls いまのフォルダの中にある
 ファイルやフォルダの名前を見る。

文字で指示 → 実行 → 結果

端末は、文字を入力・表示する窓口。
\$ は入力待ちの印で、自分では入力しない。

GUI で見た同じフォルダを、**文字で調べられる。**

CLI は、どんな作業で役立つ？

文字の指示は、手順として残し、繰り返し使える。

01 まとめて処理する

12個の文書をバックアップ

一つの指示



12個に適用

02 離れた機器を操作する

研究室のサーバーを、自分のPCから

手元の端末



サーバー

SSH などで接続

03 道具を組み合わせる

出席記録から、欠席の件数を調べる

出席記録



欠席の行を選ぶ



数える

繰り返す作業や、同じ手順を共有したい場面で役立つ。

同じ指示で、文書をまとめてコピー

12個の文書をバックアップ

同じ操作を、まとめて頼む。
次の日も、この一行を使える。

文字で残せる、コピーの指示

```
cp *.txt backup/
```

cp : コピーする

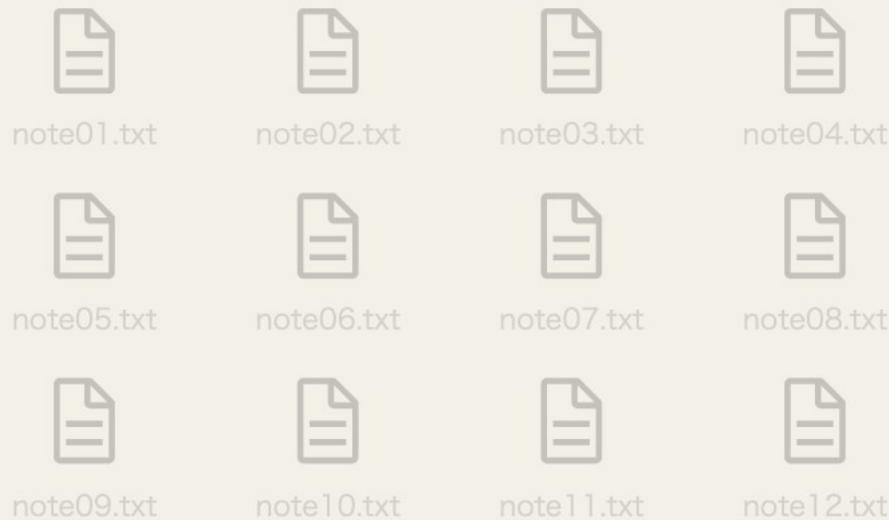
***.txt** : 名前が .txt で終わるファイル

backup/ : 用意しておいた保存先

このページは模擬操作。実際のファイルは変更しない。
指示を保存すれば、**繰り返す・再実行する・共有する**。

📁 **backup**

0 / 12



コピー前：保存先は空

まとめてコピー →

現在地を確かめて、フォルダを操作する

1 **pwd**

現在地を表示する

2 **ls**

中の名前を一覧表示する

3 **cd course**

course の中へ移動する

4 **mkdir backup**

backup フォルダを作る

文字で操作しても、動かしているのは同じフォルダ。

実行したコマンドと結果

模擬端末

まずは、現在地を確かめよう。

現在地 `/home/student`

中にあるもの **course club**

cd・mkdir は、成功しても文字を表示しないことがある。

最初

pwd を実行 →

コマンドの基本構造

ls

何をする？

名前を一覧表示

コマンド名

☐ 詳しく表示する (-l)

対象のフォルダ

現在地は student。間を空白で区切る。

命令名に、**表示の指定と対象**を添える。

—

どう表示する？

標準の形式で

オプション

course

何を対象に？

このフォルダの中

引数

```
$ ls course
```

```
archive
```

```
report.txt
```

道順は、出発点で書き方が変わる

フォルダの階層



現在地 / pwd

/home/student

絶対パス：一番上の / から

/home/student/course/report.txt

相対パス：現在地から

course/report.txt

.. は、一つ上のフォルダ

現在地が変わると、相対パスは変わる

二つの書き方で指しているのは、**同じ report.txt**

入力したコマンドは、どう実行される？

☒ 詳しい一覧 (-l)

授業用の模擬端末

 端末 / /home/student

```
$ ls -l course
```

入力した文字列

STEP 1

入力を受け取る

端末は文字を入力・表示する窓口

ls

何をする

-l

どう表示

course

何を対象に

1 端末



2 シェル



3 ls と OS



4 端末に結果

シェルは、コマンドを解釈して実行するプログラム。
この例では ls を起動し、ls が OS に処理を頼む。

最初

戻る

再生

1/4

次へ

シェルに入力して、フォルダを操作しよう

 student@course 授業用の模擬シェル

文字で、同じフォルダを操作しよう。
下の \$ の後に入力し、Enter で実行。

`pwd → ls → cd course → cat report.txt`

\$ ここに入力（例：pwd）

↑ ↓ : 入力履歴 Tab : 名前を補完 Shift+Tab : 入力欄を出る

実行

現在地 `/home/student`

```
/
├── home
└── student ← 現在地
    ├── club
    │   └── report.txt
    ├── course
    │   ├── archive
    │   └── report.txt
```

試す順番 選ぶと入力欄に入る

`pwd`

1. 現在地を見る

`ls`

2. 中の名前を見る

`cd course`

3. 授業用フォルダへ

`cat report.txt`

4. 内容を読む

`mkdir backup`

5. フォルダを作る

`cd ..`

6. 一つ上へ戻る

最初に戻る

まず `pwd` で、現在地を確かめよう。

作業に合わせて、GUI と CLI を選ぶ



GUI：画面の対象を選ぶ

選択肢を見つめながら操作したい。

写真を見比べて、一枚を選ぶ

見た目を確かめながら選べる。

アイコンやメニューが、操作の手がかり



CLI：文字で手順を指定する

同じ処理を、まとめて繰り返したい。

毎日の文書を、同じ手順で保存する

命令を保存して、再実行・共有できる。

命令の名前と書き方を、少しずつ覚える

演習4で試す 現在地を確かめる → 中を見る → 移動する → フォルダを作る

目的に合わせて、GUI と CLI を組み合わせて使える。

OSの歴史と現在

資源を分け、機器につなぎ、作業を守る。それが OS の役割である。

なぜ、今の OS の形になった？



一つずつ処理

使い方が変わる



同時に多くの作業

01 OSの役割

02 CPU

03 メモリ

04 ファイル

05 操作

06 **歴史**

初期のOSは、何を解決した？

初期の代表例

1956

GM-NAA I/O

IBM 704向けに開発
General Motors と
North American Aviation の共同開発

Robert L. Patrick

Owen Mock

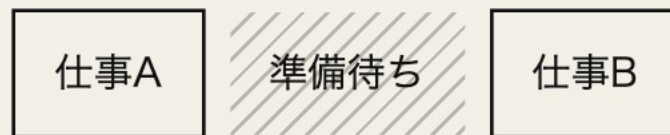
らが設計・開発に参加

一人の発明ではなく、複数の組織と技術の積み重ね

[出典：Robert L. Patrickの回想 / Computer History Museum](#)

次の仕事への切替を自動化する

人が準備する



OSが次へ進める



高価な計算機の待ち時間を減らし

一日に処理できる仕事を増やす

仕事をまとめて提出し、結果を待つ



IBM 1401 と利用者（1959年頃）

[写真・歴史資料：IBM Archives / IBM 1401](#)

カードで手順とデータを渡す

仕事をまとめて提出し、機械が処理する順番を待つ

結果は、処理後に受け取る

高価な機械を止めずに使うことを重視

バッチ処理

対話の速さより、まとめて処理する量

一台を共有して、対話しながら使う



PDP-1 と Ed Fredkin (1960年頃)

写真 : [Computer History Museum · 102631244](https://www.computerhistory.org/collections/object/102631244) 写真は開発に関わった人物と実機、複数人の同時利用風景ではない

端末から操作して、応答を待つ

複数の利用者が、一台の計算機へ要求を出す

短い時間で利用者を切り替える

待たせすぎず、他人のデータも守る必要がある

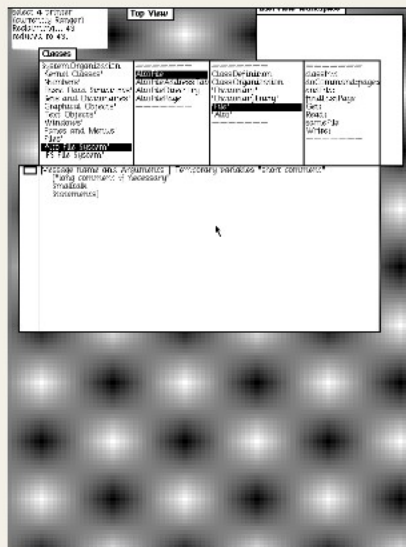
タイムシェアリング

CPU の配分・メモリ保護・所有者の管理へ

文字だけでなく、画面の対象を操作する



Xerox Alto (1973年に登場)



Smalltalk の画面

文字を入力するだけでなく
画面の対象を選んで操作

マウス、ウィンドウ、画面上の文書が、日常の
操作につながった

1968年の Engelbart のデモなど、複数の研究の積
み重ね

[Alto 写真 : Maksym Kozlenko](#) · CC BY-SA 4.0 / [画面 : Damien.b](#) · CC BY 4.0 (2024年公開の画面画像)

[CHM : GUI の歩み](#) / [1968年のデモ映像](#)

設計目標は広がった

PERSONAL COMPUTER

一人で使う

画面で操作しやすくする。
プリンタなどをつないで使う。

MOBILE

持ち歩く

電池を長持ちさせる。
位置情報や写真へのアクセスを
守る。

CLOUD

大量に分ける

多くのサービスを分けて動かす。
故障しても、影響を小さくす
る。

UNIX から広がった考え方



Dennis Ritchie (立位) と Ken Thompson

[写真 : Bell Labs archive · © 2002 Lucent Technologies](#)

UNIX · Bell Labs · 1969-

共同利用する OS と、その後に広がる設計・操作の考え方

Linux

UNIXの考え方を参考に、独立して開発

macOS / iOS と Android

共通の技術を土台にして、用途に合わせて発展した

OSの系統を、根元からたどる

UNIX と、その流れをくむ OS



系統（けいとう）は、発展のつながり。身近な OS に絞った模式図。

Windows の系統



根元から、枝をたどる ↑

サーバーでも、共有と保護が必要



通販



動画配信



チャット

別々のサービスを、同じサーバーで動かす



共有しながら、分けて管理する

使える資源や見える範囲を、サービスごとに制限

その仕組みの一例が**コンテナ**

OSの基本課題は同じ：資源を分ける・使い方をそろえる・互いを守る

オープンソースの力



成果が、次の人の出発点になる

Linux

OSの共通基盤

PyTorch

AIを学習させる共通の道具

Transformers

公開モデルを試す・応用する

利用・改変・再配布の自由が、**知識と技術の蓄積を加速する**
単に無料という意味ではなく、ライセンスがその自由を支える

コンピュータとAIの 急速な進歩を支える基盤

❏ ゼロから作り直さない

公開されたOSやAIの道具を再利用する

👤 世界中で改良する

研究者・企業・個人が、同じ成果を育てる

🔗 次の発見を早める

改良を共有し、新しい研究や製品につなぐ

[オープンソースの定義](#)

日常の操作を、OSの仕組みで説明する

レポート編集

教材ブラウザ

ファイル操作



音楽を聴きながら
レポートを保存した

この裏側で、OSは何をした？

OSの動きを確かめる →



CPU・メモリ・ファイル・保護を
一つの操作につなげて考える



演習では、実際に確かめる

```
ls -l report.txt
```

ファイルの存在・大きさ・アクセス権を確認

次回：AIと未来のコンピュータ

ここまで 回路・CPU・メモリ・OSで、プログラムが動く仕組みを学んだ

AI

文字や画像を、どう見分ける？

省電力な計算

速さと電力を、どう両立する？

量子コンピュータ

量子の性質を、どう計算に使う？

次回の入口 コンピュータは、手書きの「9」をどう読み取る？