Distributed Programs

Concurrent Processes and Concurrent Objects

国立研究開発法人 情報通信研究機構
慶應義塾大学名誉教授
徳田英幸

ちょっと復習。。。

What is an Autonomous Distributed Cooperative System (ADC System)？

- No supervisor which can control/manage the entire system
  システム内にシステム全体を制御／統治するスパーバイザは存在しない。
- The system consists of autonomous, distributed and cooperative subsystems 各サブシステムは、自律、分散した構成要素からなる。
- The system functions are realized by cooperations among subsystems
  全体のシステムの機能は、サブシステム間の協調作業によって遂行される。

演習-1 (4/16/2018)

- 集中型システムをリストせよ
  - 事例-Ca, Cb, Cc
- 分散型システムをリストせよ
  - 事例-Da, Db, Dc
  - Da-Dcの中で、各構成要素が自律性をもっているものに〇を付与せよ
  - Da-Dcの中で、各構成要素が自律性とかつ協調性を持っている場合は、◎を付与せよ

情報システムを考える際に

- 何に着目するか？
  - 集中型／分散型／ハイブリッド型
  - システム全体の性能は？
    - 定性的評価／定量的評価
    - ボトルネックはどこか？
  - 制御の流れ
    - 単一故障から全体故障へ
      - Single point of failure　vs. graceful degradation
  - データの流れ
  - メッセージの流れ

自律分散協調システムの目的

- 機能拡大
- コスト性能比の改善
- 分散処理による効率・サービスの改善
- オンラインリアルタイム処理の実現
- 局所化による通信量の低減
- 構成要素のmモジュール化
- 拡張性の保証
- 集団組織の効率化
- 信頼性・耐故障性の改善
- 状況・環境変化への適応
- 生存可能性の増大

Distributed Programs
Concurrent Processes and Concurrent Objects

慶應義塾大学環境情報学部
徳田英幸

ADC Models in Information Systems

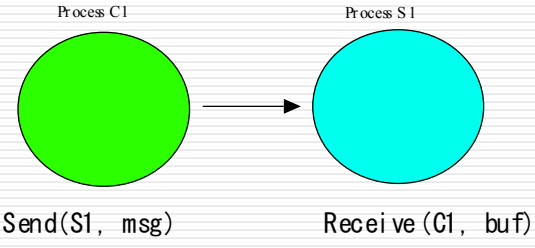
- System Architecture システムアーキテクチャ的な見方
- Distributed Program 分散プログラム的な見方
- Basic Concepts
  - What are system components? 構成要素は何か？
  - How to cooperate?それらがどのように協調作業を進めるのか？
    - Control Flow 制御の流れ
    - Data Flow データの流れ
    - Message Flow メッセージの流れ

Distributed Program Model 分散プログラムのモデル

- Concurrent Process Model 並行プロセスモデル
  - Program = a set of cooperating (sequential) processes
- Concurrent Object Model 並行オブジェクトモデル
  - Program = a set of cooperating objects
- 対象としているシステムのモデル化に適している。
- いくつかの分散・並行プログラミング言語も開発されている。

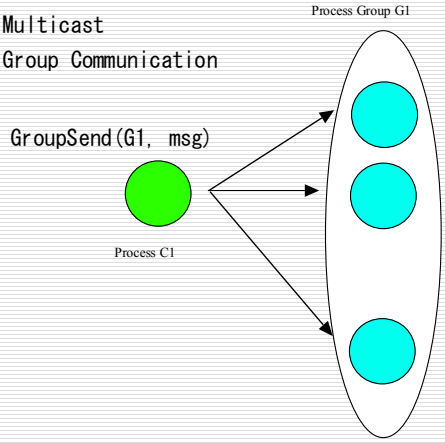
Concurrent Process: Interprocess Comm. (1)

・Unicast



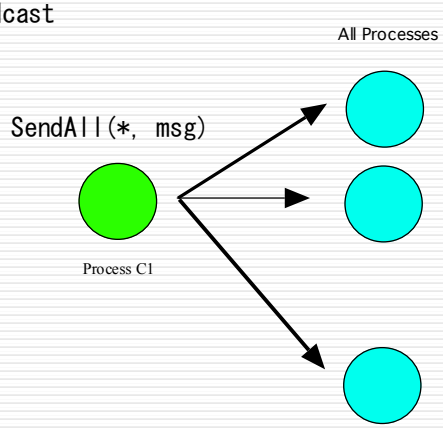
Concurrent Process: Interprocess Comm. (2)

- Multicast
- Group Communication

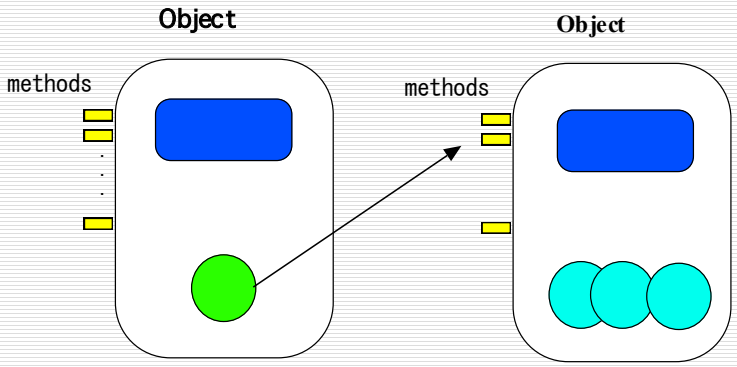


Concurrent Process: Interprocess Comm. (3)

- Broadcast



Concurrent Objects 並行オブジェクト



Concurrent processes and their description

並行プロセスの挙動とその記述

慶應義塾大学環境情報学部
徳田英幸

プロセス間通信: Direct IPC

- The communication is done between two processes without using any communication entity.

Bsend(process_id, message, size)
Nsend(process_id, message, size)

Brec(process_id, buffer, size)
Nrec(process_id, buffer, size)

process_id = Brecany(buffer, size)
process_id = Nrecany(buffer, size)

Request(process_id, message, sizeof(message), buffer, sizeof(buf)

Reply(process_id, message, size)

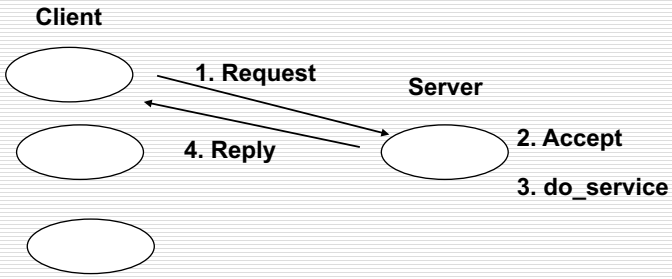
プロセス間通信: Direct IPC

- **特定の相手との送受信**
- **Bsend(process_id, message, size) /* Blocking Send**
- **Nsend(process_id, message, size) /* Non-Blocking Send**
- **Brec(process_id, buffer, size) /* Blocking Receive**
- **Nrec(process_id, buffer, size) /* Non-Blocking Receive**

- **/* 不特定の相手からの受信**
- **process_id = Brecany(buffer, size)**
- **process_id = Nrecany(buffer, size)**
- /* サーバの要求の送信 （Nsend+Brec）
- **Request(process_id, message, sizeof(message), buffer, sizeof(buf)**
- /* クライアントへの返信
- **Reply(process_id, message, size)**

Client/Server Model

- Client/Server Model is a well-known model.
- A client (i.e., a user process) sends the request message to a server process, which then does he work and sends back the results.



Actions in a Server

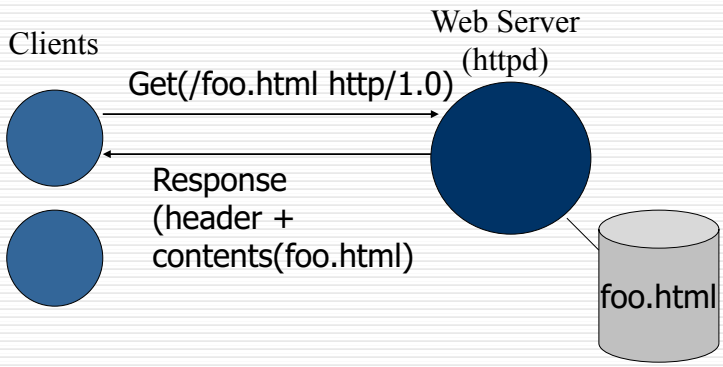
```
Process Server() {
    init();
    while(1) {
        pid = Brecany(..) /* リクエスト待ち */

        do_service(pid, …) /* 処理 */

        reply(pid, …)          /* 結果を返信 */
    }
}
```

Web Server's Architecture

Client/Server in Web



Actions in a Simple Web Server

```
Process web_server() {
    init(); /* 初期化 */
    while(1) { /* 繰り返し処理  */
        pid = Brecany(..)

        do_service(pid, …)

        reply(pid, …)
    }
}
```


TelnetによるHttpdアクセス

- telnet 133.27.246.236 80
- GET index.html HTTP/1.0
  - HTTP/1.1 400 Bad Request
- telnet www.ht.sfc.keio.ac.jp 80
- GET /index.html HTTP/1.0

HTTP/1.1のメソッド

- <method> <URL> <HTTP/version>
- GET　　URLのヘッダ情報とコンテンツの取得
- HEAD　　URLのヘッダ情報の取得
- POST　　URLへ情報を送る(CGIで使用)
- PUT　　URLにデータを保存
- DELETE　URLの削除
- TRACE　ループバック容
- OPTIONS 機能の一覧

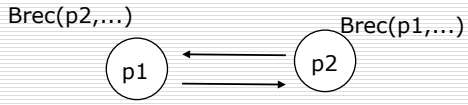
HTTP/1.1の主なリターンコード

- <HTTP version> <RETCODE status>
- 200 OK
- 206 Partial Content
- 301 Moved Permanently
- 304 Not Modified
- 403 Forbidden
- 404 Not Found
- 408 Request Timeout
- 500 Internal Server Error

並行プロセスの挙動

Deadlock (1)

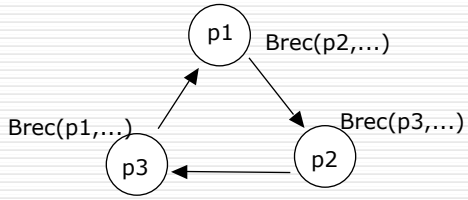
```
Process P1                    Process P2
{                             {
  …                             …
  brec(p2, …);                  brec(p1, …);
  …                             …
}                             }
```



並行プロセスの挙動

What is deadlock？

- システムを構成しているすべてのプロセスが起きるはずのない事象を待ち続ける状態

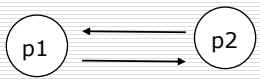


並行プロセスの挙動

Deadlock（2）

```
Process P1
{
  …
  request(printer,…)
  request(tape, …)
  …
}
```

```
Process P2
{
  …
  request(tape, …)
  request(printer,…)
  …
}
```

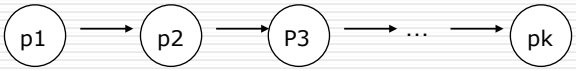


並行プロセスの挙動

Pipeline（1）

```
Process P1
{
  …
  bsend(p2, …);
  …
}
```

```
Process P2
{
  …
  brec(p1, …);
  do_my_part();
  bsend(p3, …);
  …
}
```

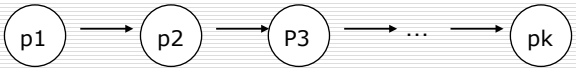


並行プロセスの挙動

Pipeline（2）

```
Process P1
{
  …
  Nsend(p2, …);
  …
}
```

```
Process P2
{
  …
  Nsend(p3, …);
  …
}
```



並行プロセスの挙動

Worker Model

```
Server S1
{
  …
  Brecany(msg,…)
    do_service(...)
  reply(msg, …)
  …
}
```

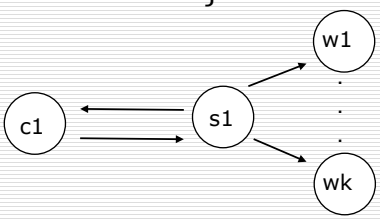
並行プロセスの挙動

Dynamic Worker Model

```
Server S1
{
  …
    ci=Brecany(msg,…)
    create_worker(ci,...)
  …
}
```

```
Process w1
{
  …
  do_service(msg,…)
  reply(ci,…)

}
```



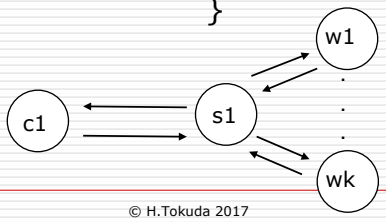
並行プロセスの挙動

Static Worker Model

```
Server S1
{
  …
    ci=Brecany(msg,…)
    if(ci is worker)
       assign_task(ci, …)
    else
       enque(ci, task)
  …
}
```

```
Process w1
{
  …
  request(s1, ...)
  do_service(msg,…)f
  reply(ci,…)

}
```



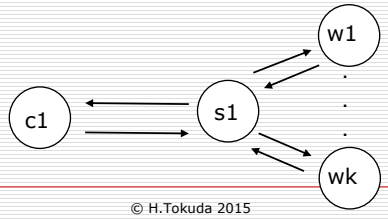
並行プロセスの挙動

Static Worker Model

```
Server S1
{
  …
    ci=Brecany(msg,…)
    if(ci is worker) {
        if (TaskQ is not empy) {
            assign_task(ci, deq(TaskQ), …)
        }else{
            enque(WorkerQ, ci)
        }
     }else{ // ci is client
        if (WorkerQ is not empty) {
            assign_task(deq(WorkerQ), ci, …)
        }else{
            enque(TaskQ, ci)
        }
     }
    ….
}
```

```
Process w1
{
  …
  request(s1, …)
  do_service(msg,…)
  reply(ci,…)

}
```



課題-1: Dist. Partitioned Sort

- 1からNまでの整数の分散ソート
  - Distributed Sort for the numbers between 1 and N
  - Initial value, 100 cards, N=1,000
  - Processes: P1- P10
  - 各プロセスは、P1から順に横一列に並んでおり、自分の隣接しているプロセスとのみ交信出来るものと仮定する。
- 問題
  - P1からP10までの各プロセスにランダムに選ばれた10個の整数が初期の値として与えられた時、どのようにしてソートすることができるか？日本語でのアルゴリズムの記述と以下の擬似言語とDirect IPCプリミティブを使って記述せよ。
  - 各プロセスは、どのような条件で、ソートが終了したことを判定できるかを述べよ。

課題-1 Dist. Partitioned Sort

- 仮定

  各プロセスは、自分のpidと協調しているプロセスの総数は、与えられて
  いるとする。
  右側のプロセスのpidは、pid=succ(i), 左側のプロセスのpidは、
  Pid =pred(i)で特定できるとする。

```
Process Pi(pid i, int m, int card[10])
{
        init();
        while (1) {
          …
          do_exchange( );
         …
          if (termination_condition() == true) exit;
        }
}
```