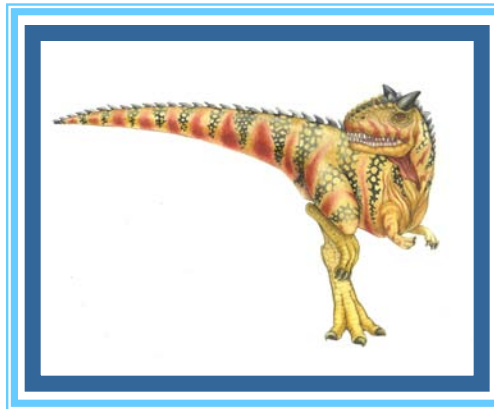


Chapter 3: Processes





Chapter 3: Processes

- Process Concept
- Process Scheduling
- Operations on Processes
- Interprocess Communication
- Examples of IPC Systems
- Communication in Client-Server Systems





Objectives

- To introduce the notion of a process -- a program in execution, which forms the basis of all computation
- To describe the various features of processes, including scheduling, creation and termination, and communication
- To explore interprocess communication using shared memory and message passing
- To describe communication in client-server systems





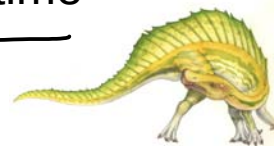
Process Concept

process: program in execution

- An operating system executes a variety of programs:
 - Batch system – **jobs**
 - Time-shared systems – **user programs** or **tasks**
- Textbook uses the terms **job** and **process** almost interchangeably
- **Process** – a program in execution; process execution must progress in sequential fashion
- Multiple parts
 - The program code, also called **text section**
 - Current activity including program counter, processor registers
 - **Stack** containing temporary data
 - ▶ Function parameters, return addresses, local variables
 - **Data section** containing global variables
 - **Heap** containing memory dynamically allocated during run time

instr. pointer
=
program counter

malloc (N)
↳ user





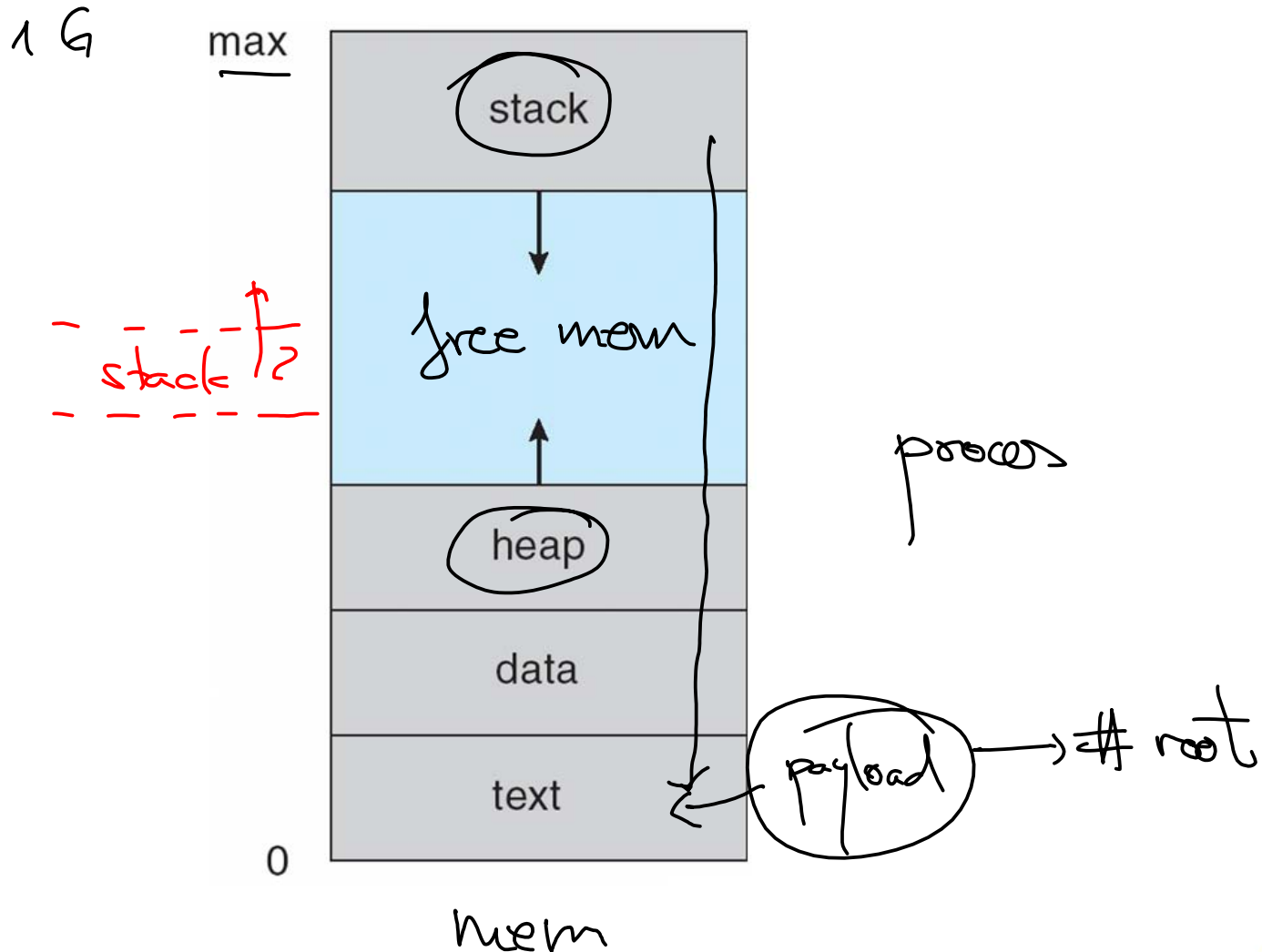
Process Concept (Cont.)

- Program is ***passive*** entity stored on disk (**executable file**), process is ***active***
 - Program becomes process when executable file loaded into memory
- Execution of program started via GUI mouse clicks, command line entry of its name, etc
- One program can be several processes
 - Consider multiple users executing the same program





Process in Memory





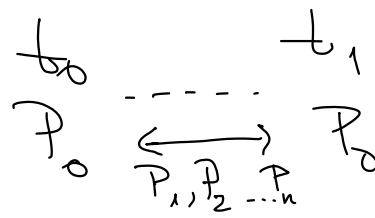
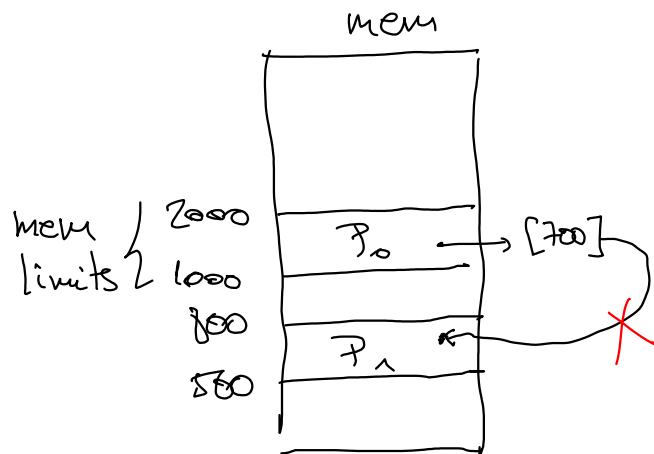
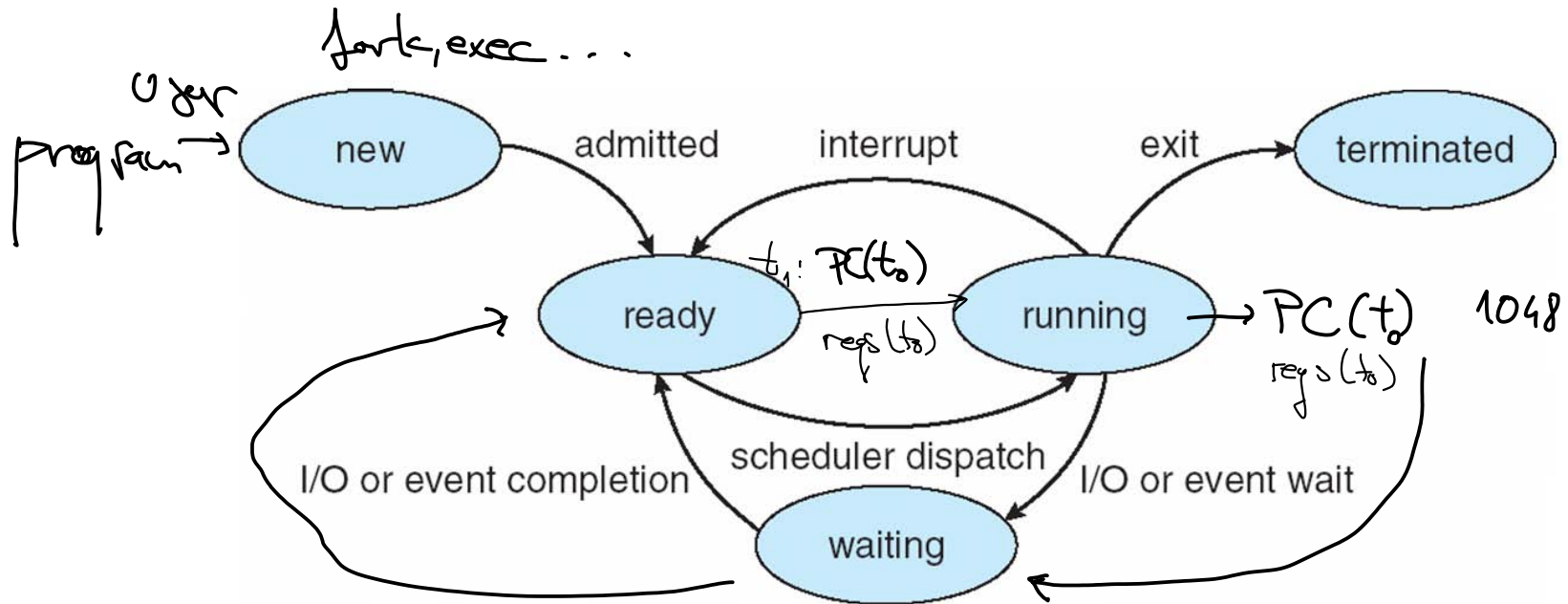
Process State

- As a process executes, it changes **state**
 - **new**: The process is being created
 - **running**: Instructions are being executed
 - **waiting**: The process is waiting for some event to occur
 - **ready**: The process is waiting to be assigned to a processor
 - **terminated**: The process has finished execution





Diagram of Process State



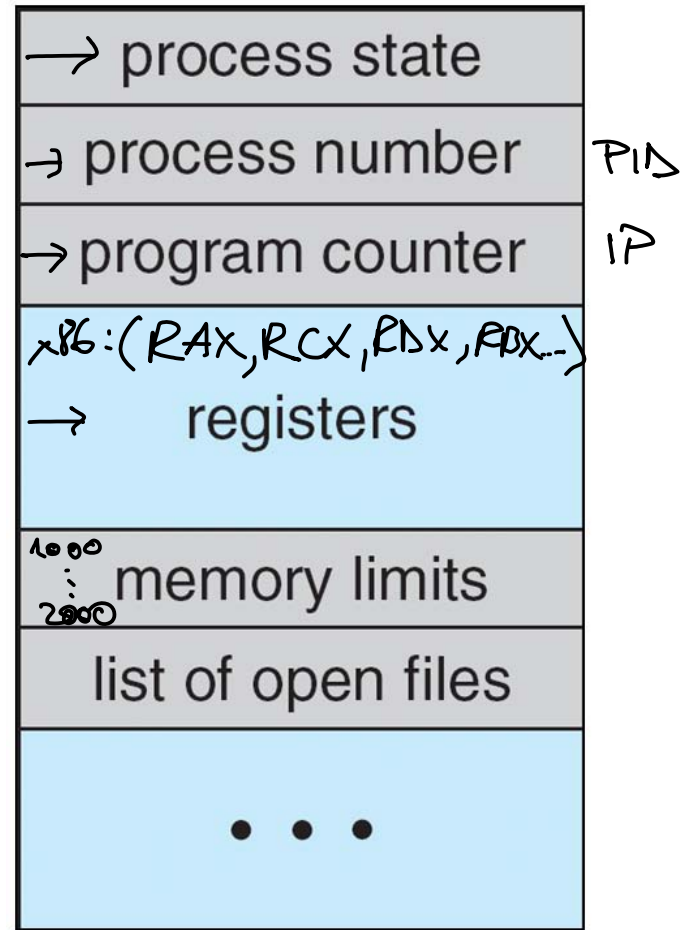


Process Control Block (PCB)

Information associated with each process
(also called **task control block**)

- Process state – running, waiting, etc
- Program counter – location of instruction to next execute
- CPU registers – contents of all process-centric registers
- CPU scheduling information- priorities, scheduling queue pointers
- Memory-management information – memory allocated to the process
- Accounting information – CPU used, clock time elapsed since start, time limits
- I/O status information – I/O devices allocated to process, list of open files

struct h

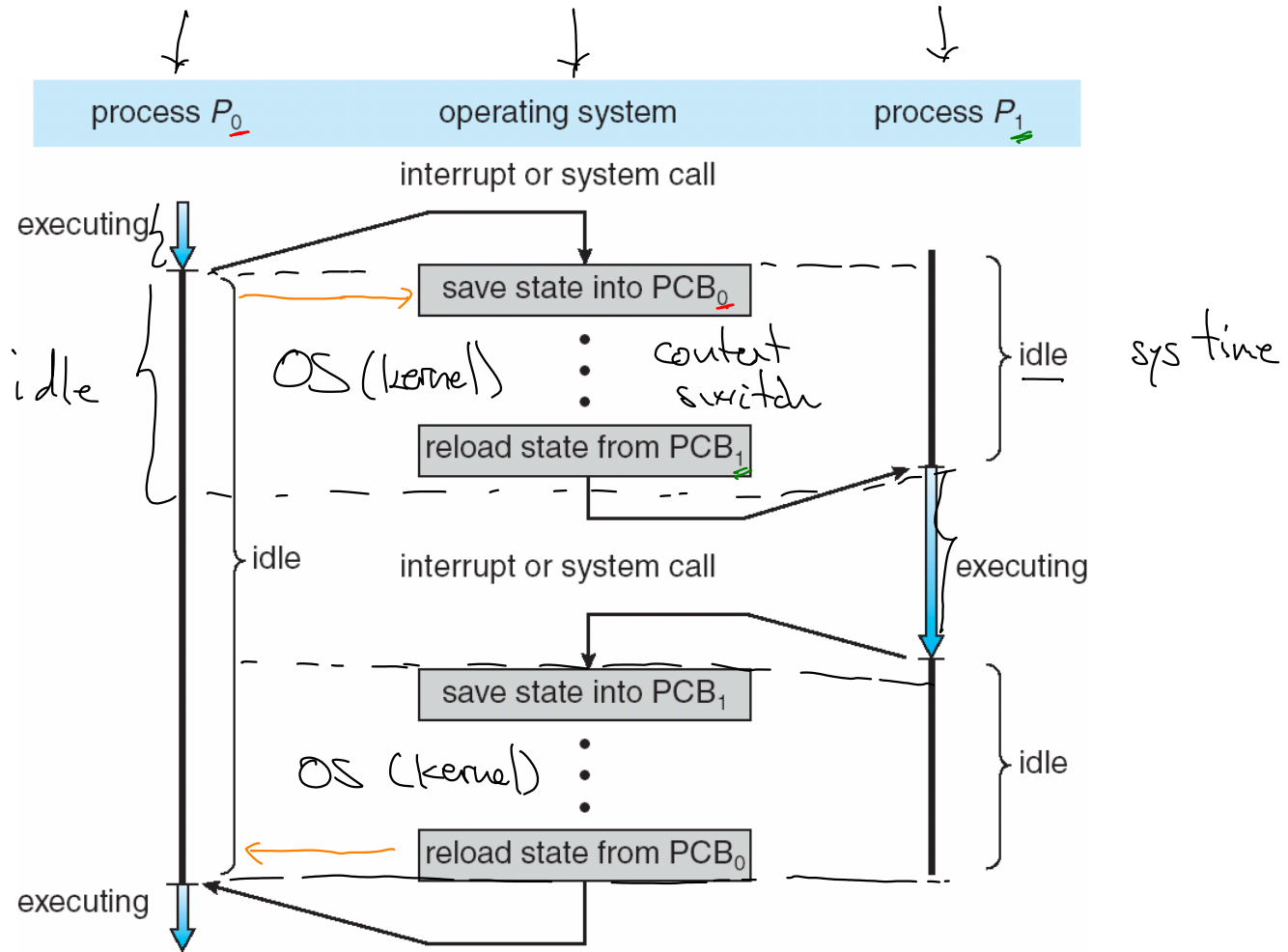


{ PCB;





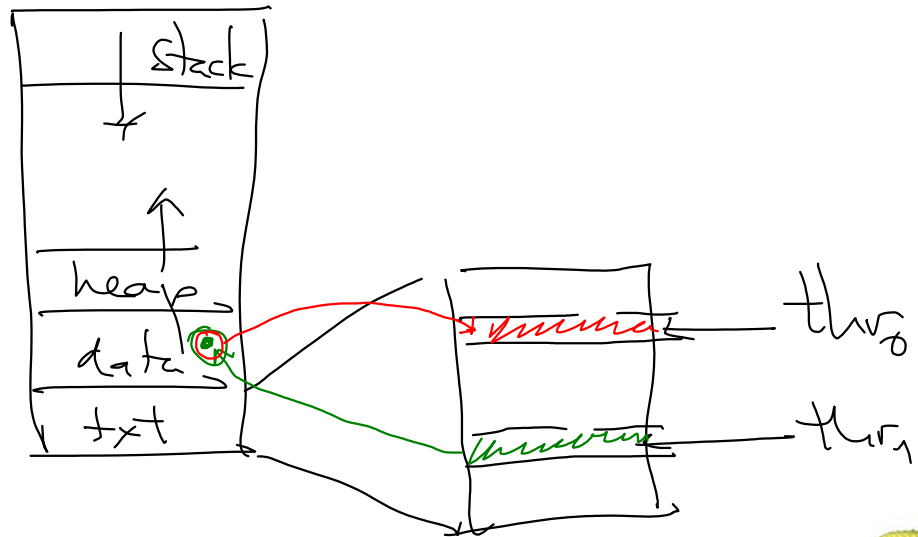
CPU Switch From Process to Process





Threads

- So far, process has a single thread of execution
- Consider having multiple program counters per process
 - Multiple locations can execute at once
 - ▶ Multiple threads of control -> **threads**
- Must then have storage for thread details, multiple program counters in PCB
- See next chapter

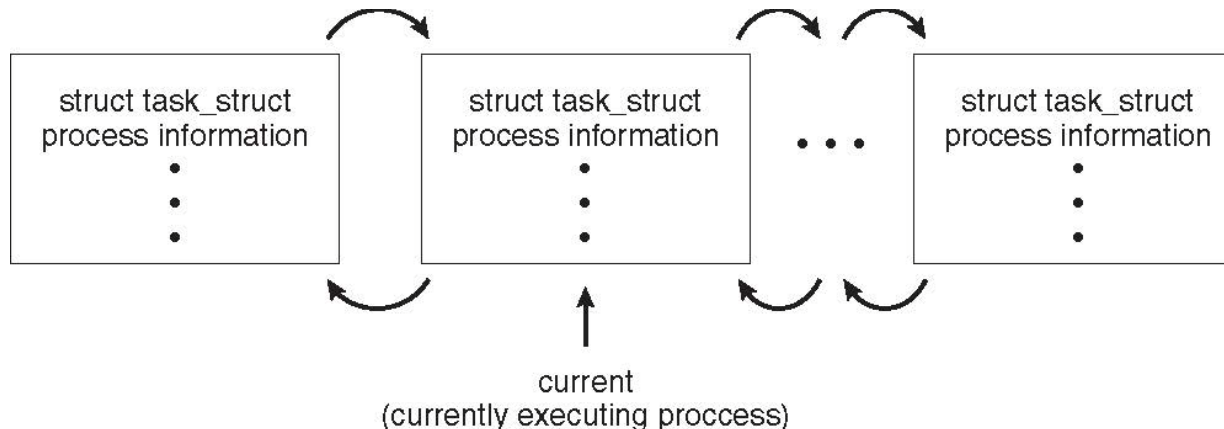




Process Representation in Linux

Represented by the C structure `task_struct`

```
pid t_pid; /* process identifier */
long state; /* state of the process */
unsigned int time_slice /* scheduling information */
struct task_struct *parent; /* this process's parent */
struct list_head children; /* this process's children */
struct files_struct *files; /* list of open files */
struct mm_struct *mm; /* address space of this process */
```





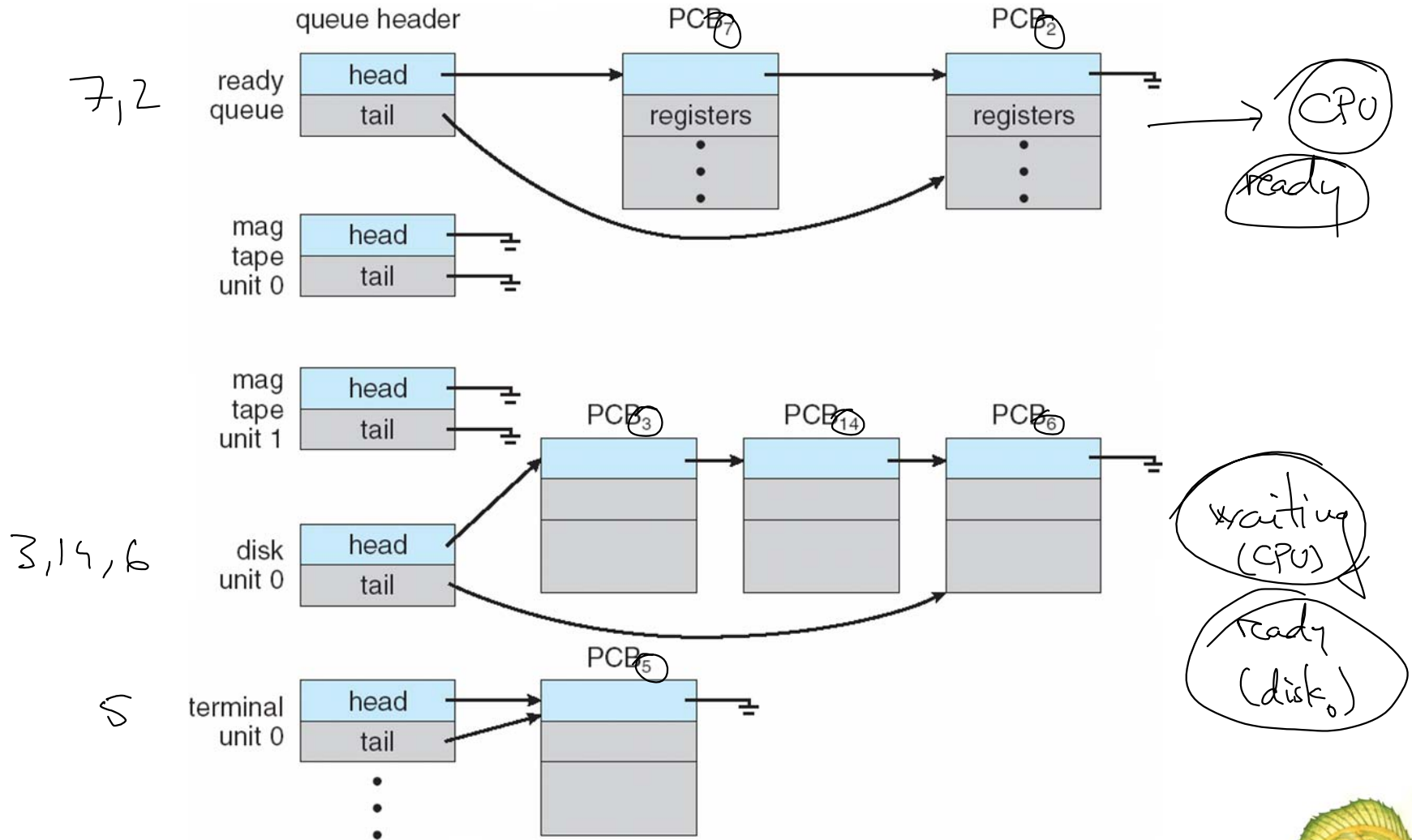
Process Scheduling

- Maximize CPU use, quickly switch processes onto CPU for time sharing
- **Process scheduler** selects among available processes for next execution on CPU
- Maintains **scheduling queues** of processes
 - **Job queue** – set of all processes in the system (all procs)
 - **Ready queue** – set of all processes residing in main memory, ready and waiting to execute
 - **Device queues** – set of processes waiting for an I/O device
 - Processes migrate among the various queues





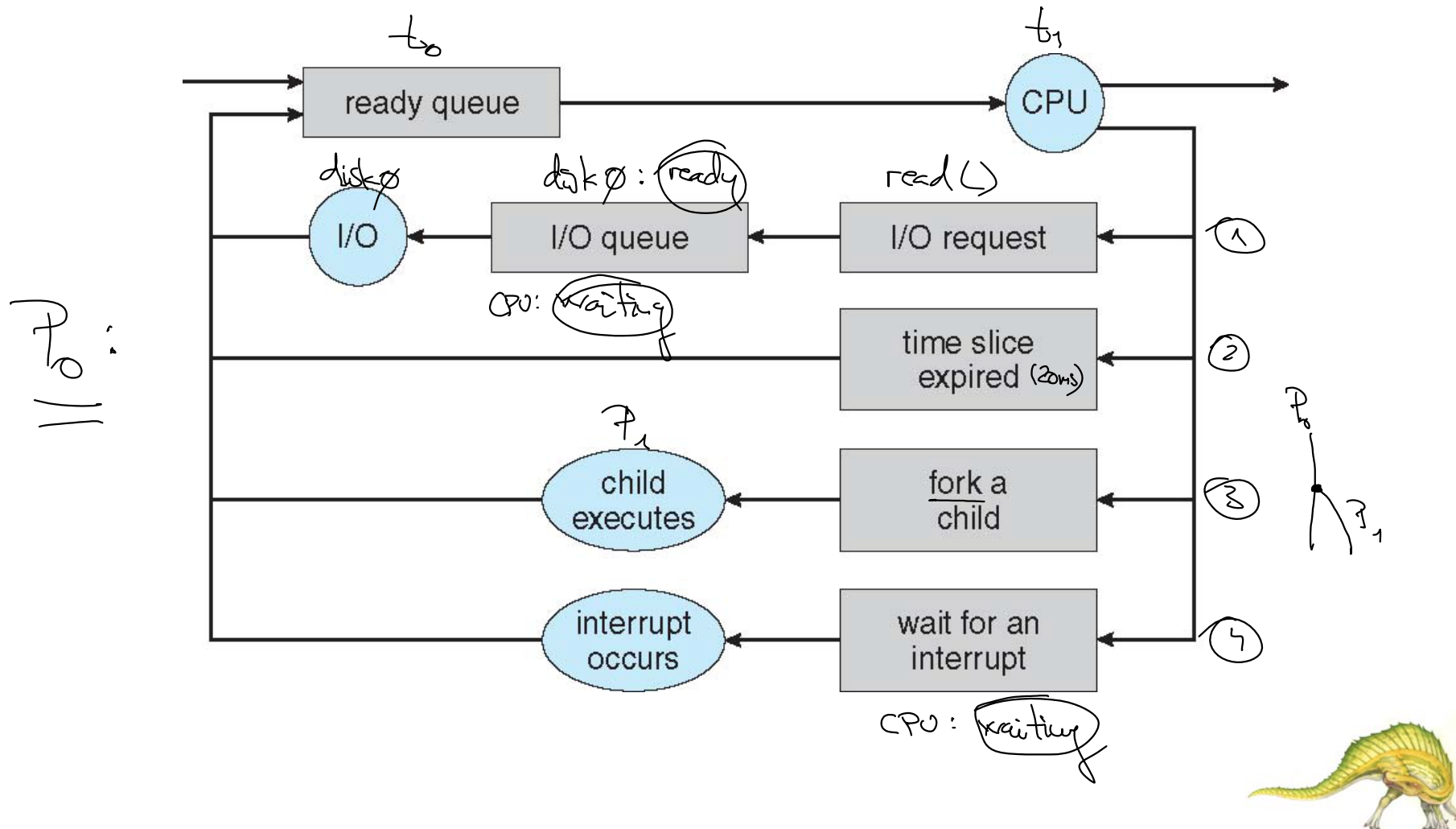
Ready Queue And Various I/O Device Queues





Representation of Process Scheduling

- **Queueing diagram** represents queues, resources, flows





Schedulers

- **Short-term scheduler** (or **CPU scheduler**) – selects which process should be executed next and allocates CPU
 - Sometimes the only scheduler in a system
 - Short-term scheduler is invoked frequently (milliseconds) \Rightarrow (must be fast)
- **Long-term scheduler** (or **job scheduler**) – selects which processes should be brought into the ready queue
 - Long-term scheduler is invoked infrequently (seconds, minutes) \Rightarrow (may be slow)
 - The long-term scheduler controls the **degree of multiprogramming**
- Processes can be described as either:
 - **I/O-bound process** – spends more time doing I/O than computations, many short CPU bursts
 - **CPU-bound process** – spends more time doing computations; few very long CPU bursts
- Long-term scheduler strives for good **process mix**

myOp(1)

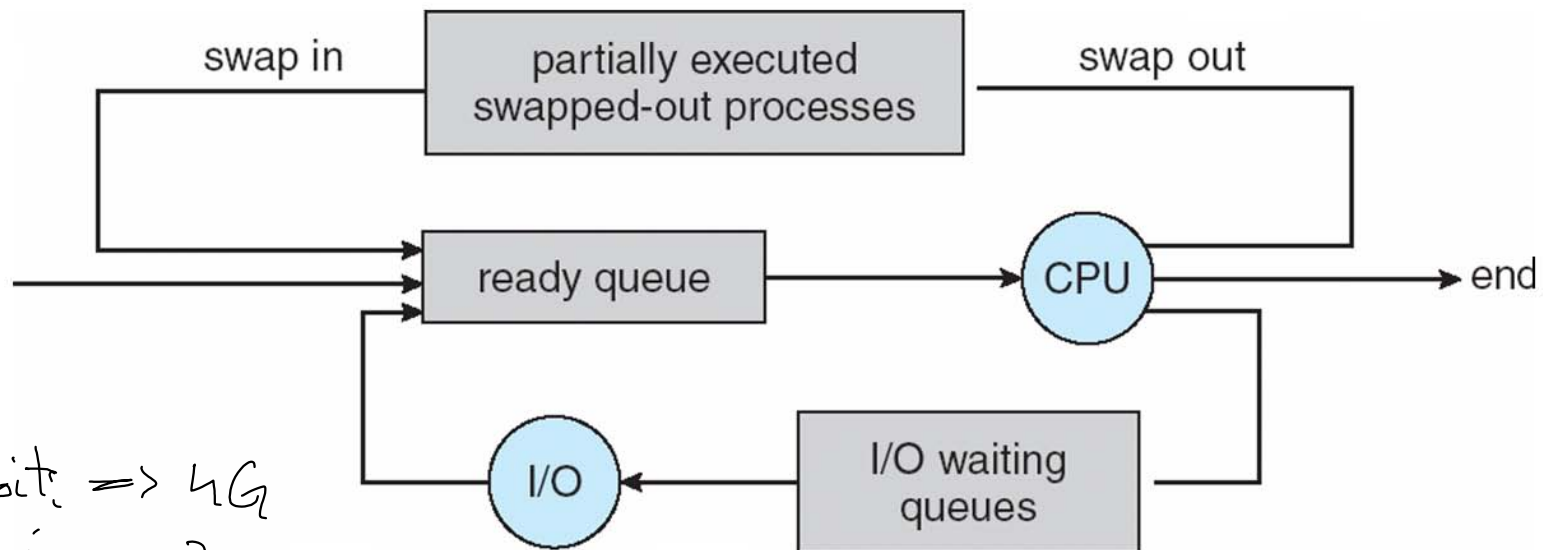
numpy
scipy





Addition of Medium Term Scheduling

- **Medium-term scheduler** can be added if degree of multiple programming needs to decrease
 - Remove process from memory, store on disk, bring back in from disk to continue execution: **swapping**



32-bit \Rightarrow 4G
64-bit \Rightarrow ?





Multitasking in Mobile Systems

- Some mobile systems (e.g., early version of iOS) allow only one process to run, others suspended
- Due to screen real estate, user interface limits iOS provides for a
 - Single **foreground** process- controlled via user interface
 - Multiple **background** processes– in memory, running, but not on the display, and with limits
 - Limits include single, short task, receiving notification of events, specific long-running tasks like audio playback
- Android runs foreground and background, with fewer limits
 - Background process uses a **service** to perform tasks
 - Service can keep running even if background process is suspended
 - Service has no user interface, small memory use





Context Switch

- When CPU switches to another process, the system must **save the state** of the old process and load the **saved state** for the new process via a **context switch**
- **Context** of a process represented in the **PCB**
- Context-switch time is overhead; the system does no useful work while switching
 - The more complex the OS and the PCB → the longer the context switch
- Time dependent on hardware support
 - Some hardware provides multiple sets of registers per CPU → multiple contexts loaded at once





Operations on Processes

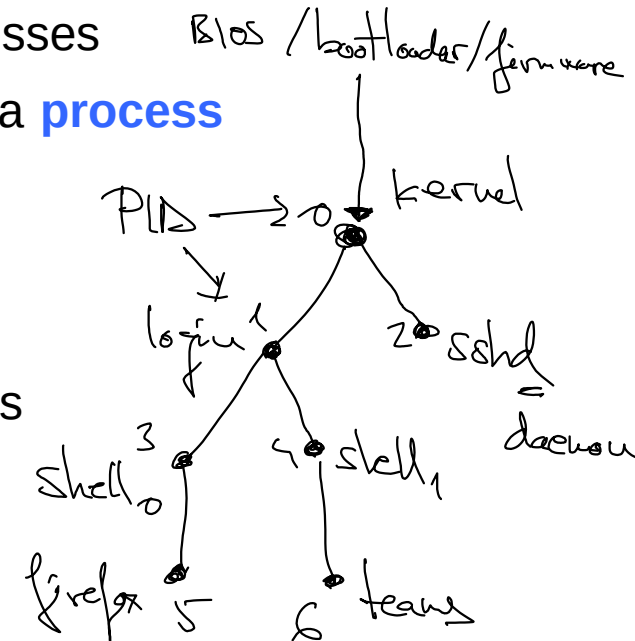
- System must provide mechanisms for:
 - process creation,
 - process termination,
 - and so on as detailed next





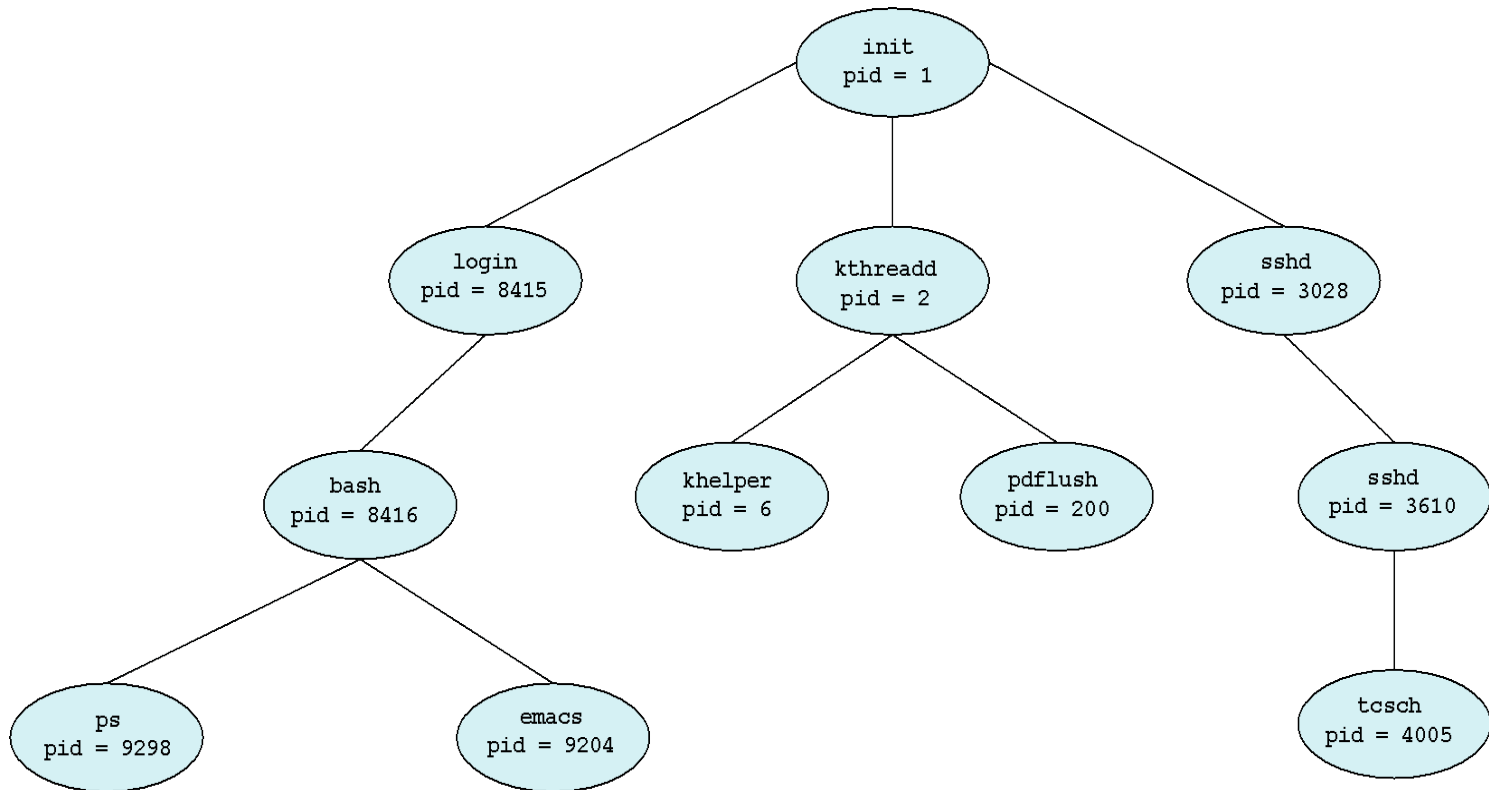
Process Creation

- **Parent** process create **children** processes, which, in turn create other processes, forming a **tree** of processes
- Generally, process identified and managed via a **process identifier (pid)**
- Resource sharing options
 - Parent and children share all resources
 - Children share subset of parent's resources
 - Parent and child share no resources
- Execution options
 - Parent and children execute concurrently
 - Parent waits until children terminate





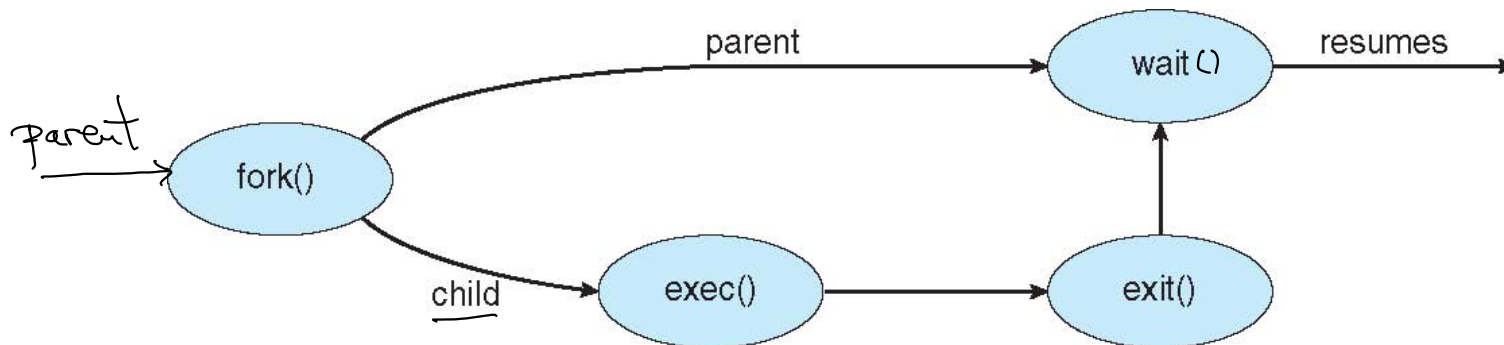
A Tree of Processes in Linux





Process Creation (Cont.)

- Address space
 - Child duplicate of parent
 - Child has a program loaded into it
- UNIX examples
 - **fork()** system call creates new process
 - **exec()** system call used after a **fork()** to replace the process' memory space with a new program





C Program Forking Separate Process

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>

int main()
{
    → pid_t pid;

    /* fork a child process */
    → pid = fork();

    if (pid < 0) { /* error occurred */
        fprintf(stderr, "Fork Failed");
        return 1;
    }
    else if (pid == 0) { /* child process */
        execvp("/bin/ls", "ls", NULL);
    }
    else { /* parent process */
        /* parent will wait for the child to complete */
        wait(NULL);
        printf("Child Complete");
    }
    return 0;
}

→ wait(&child-ret)
```

fork()

- 0: child
- child's PID: parent

parent + child

args are

no & wait return! main(ls) { ; ret 0

args = 2 "ls", NULL → argc = 1
main(argc, argv)





Creating a Separate Process via Windows API

```
#include <stdio.h>
#include <windows.h>

int main(VOID)
{
    → STARTUPINFO si;
    → PROCESS_INFORMATION pi;

    /* allocate memory */
    ZeroMemory(&si, sizeof(si));
    si.cb = sizeof(si);
    ZeroMemory(&pi, sizeof(pi));

    /* create child process */
    if (!CreateProcess(NULL, /* use command line */
        "C:\\WINDOWS\\system32\\mspaint.exe", /* command */ ← exec()
        NULL, /* don't inherit process handle */
        NULL, /* don't inherit thread handle */
        FALSE, /* disable handle inheritance */
        0, /* no creation flags */
        NULL, /* use parent's environment block */
        NULL, /* use parent's existing directory */
        &si,
        &pi))
    {
        fprintf(stderr, "Create Process Failed");
        return -1;
    }

    /* parent will wait for the child to complete */
    WaitForSingleObject(pi.hProcess, INFINITE);
    printf("Child Complete");

    /* close handles */
    CloseHandle(pi.hProcess);
    CloseHandle(pi.hThread);
}
```

fork() →
+
exec()

ew
crd

10 args!

wait() →





Process Termination

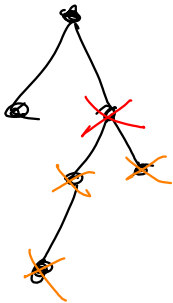
- Process executes last statement and then asks the operating system to delete it using the **exit()** system call.
 - Returns status data from child to parent (via **wait()**)
 - Process' resources are deallocated by operating system
- Parent may terminate the execution of children processes using the **abort()** system call. Some reasons for doing so:
 - Child has exceeded allocated resources
 - Task assigned to child is no longer required
 - The parent is exiting and the operating systems does not allow a child to continue if its parent terminates



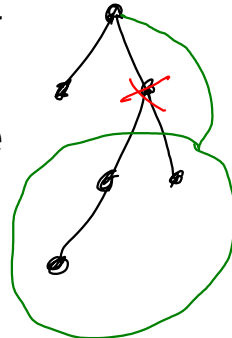


Process Termination

- Some operating systems do not allow child to exist if its parent has terminated. If a process terminates, then all its children must also be terminated.



- **cascading termination.** All children, grandchildren, etc. are terminated.
- The termination is initiated by the operating system.



- The parent process may wait for termination of a child process by using the **wait()** system call. The call returns status information and the pid of the terminated process

→ **pid = wait(&status);**

- ■ If no parent waiting (did not invoke **wait()**) process is a **zombie**

- If parent terminated without invoking **wait**, process is an **orphan**

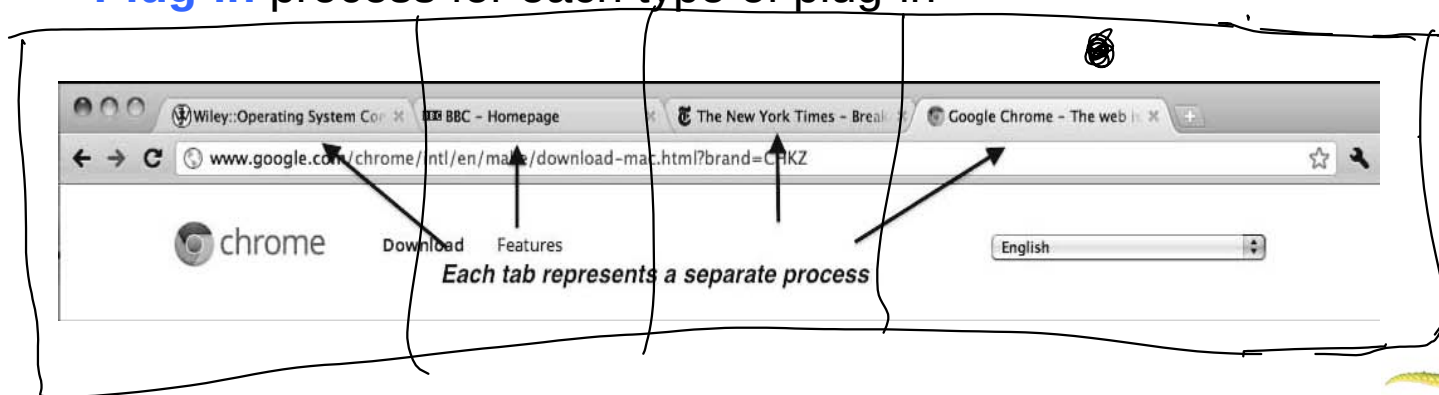
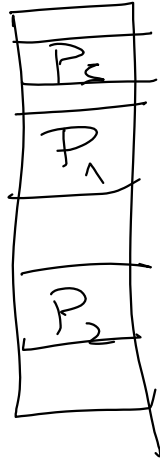
```
graph TD
    P((P)) --> C((C))
    P --> D((D))
    P -- X --> E((E))
    C -- X --> F((F))
    D --> G((G))
    E -- X --> H((H))
    F -- X --> I((I))
    G --> J((J))
    H -- X --> K((K))
    I -- X --> L((L))
    J --> M((M))
    K -- X --> N((N))
    L -- X --> O((O))
    M --> P2((P2))
    N -- X --> Q((Q))
    O -- X --> R((R))
    P2 --> S((S))
    Q -- X --> T((T))
    R -- X --> U((U))
    S --> V((V))
    T -- X --> W((W))
    U -- X --> X2((X2))
    V --> Y((Y))
    W -- X --> Z((Z))
    X2 -- X --> AA((AA))
    Y --> AB((AB))
    Z -- X --> AC((AC))
    AA -- X --> AD((AD))
    AB --> AE((AE))
    AC -- X --> AF((AF))
    AD -- X --> AG((AG))
    AE --> AH((AH))
    AF -- X --> AI((AI))
    AG -- X --> AJ((AJ))
    AH --> AK((AK))
    AI -- X --> AL((AL))
    AJ -- X --> AM((AM))
    AK --> AN((AN))
    AL -- X --> AO((AO))
    AM -- X --> AP((AP))
    AN --> AQ((AQ))
    AO -- X --> AR((AR))
    AP -- X --> AS((AS))
    AQ --> AT((AT))
    AR -- X --> AU((AU))
    AS -- X --> AV((AV))
    AT --> AW((AW))
    AU -- X --> AX((AX))
    AV -- X --> AY((AY))
    AW --> AZ((AZ))
    AX -- X --> BA((BA))
    AY -- X --> BB((BB))
    AZ --> BC((BC))
    BA -- X --> BD((BD))
    BB -- X --> BE((BE))
    BC --> BF((BF))
    BD -- X --> BG((BG))
    BE -- X --> BH((BH))
    BF --> BI((BI))
    BG -- X --> BJ((BJ))
    BH -- X --> BK((BK))
    BI --> BL((BL))
    BJ -- X --> BM((BM))
    BK -- X --> BN((BN))
    BL --> BO((BO))
    BM -- X --> BP((BP))
    BN -- X --> BQ((BQ))
    BO --> BR((BR))
    BP -- X --> BS((BS))
    BQ -- X --> BT((BT))
    BR --> BU((BU))
    BS -- X --> BV((BV))
    BT -- X --> BU2((BU2))
    BU --> BV2((BV2))
    BU2 -- X --> BW((BW))
    BV --> BX((BX))
    BV2 -- X --> BY((BY))
    BW --> BZ((BZ))
    BX --> CA((CA))
    BY -- X --> CB((CB))
    BZ --> CC((CC))
    CA --> CD((CD))
    CB -- X --> CE((CE))
    CC --> CF((CF))
    CD --> CG((CG))
    CE -- X --> CH((CH))
    CF --> CI((CI))
    CG --> CJ((CJ))
    CH -- X --> CK((CK))
    CI -- X --> CL((CL))
    CJ --> CM((CM))
    CK -- X --> CN((CN))
    CL -- X --> CO((CO))
    CM --> CP((CP))
    CN -- X --> CQ((CQ))
    CO -- X --> CR((CR))
    CP --> CS((CS))
    CQ -- X --> CT((CT))
    CR -- X --> CU((CU))
    CS --> CV((CV))
    CT -- X --> CW((CW))
    CU -- X --> CX((CX))
    CV --> CY((CY))
    CW -- X --> CZ((CZ))
    CX -- X --> DA((DA))
    CY -- X --> DB((DB))
    CZ --> DD((DD))
    DA -- X --> DE((DE))
    DB -- X --> DF((DF))
    DD --> DG((DG))
    DE -- X --> DH((DH))
    DF -- X --> DI((DI))
    DG --> DJ((DJ))
    DH -- X --> DK((DK))
    DI -- X --> DL((DL))
    DJ --> DM((DM))
    DK -- X --> DN((DN))
    DL -- X --> DO((DO))
    DM --> DP((DP))
    DN -- X --> DQ((DQ))
    DO -- X --> DR((DR))
    DP --> DS((DS))
    DQ -- X --> DT((DT))
    DR -- X --> DU((DU))
    DS --> DV((DV))
    DT -- X --> DW((DW))
    DU -- X --> DX((DX))
    DV --> DY((DY))
    DW -- X --> DZ((DZ))
    DX -- X --> EA((EA))
    DY -- X --> EB((EB))
    DZ --> EC((EC))
    EA --> ED((ED))
    EB -- X --> EE((EE))
    EC --> EF((EF))
    ED --> EG((EG))
    EE -- X --> EH((EH))
    EF --> EI((EI))
    EG --> EJ((EJ))
    EH -- X --> EK((EK))
    EI -- X --> EL((EL))
    EJ --> EM((EM))
    EK -- X --> EN((EN))
    EL -- X --> EO((EO))
    EM --> EP((EP))
    EN -- X --> EQ((EQ))
    EO -- X --> ER((ER))
    EP --> ES((ES))
    EQ -- X --> ET((ET))
    ER -- X --> EU((EU))
    ES --> EV((EV))
    ET -- X --> EW((EW))
    EU -- X --> EX((EX))
    EV --> EY((EY))
    EW -- X --> EZ((EZ))
    EX -- X --> FA((FA))
    EY -- X --> FB((FB))
    EZ --> FC((FC))
    FA --> FD((FD))
    FB -- X --> FE((FE))
    FC --> FF((FF))
    FD --> FG((FG))
    FE -- X --> FH((FH))
    FF --> FI((FI))
    FG --> FJ((FJ))
    FH -- X --> FK((FK))
    FI -- X --> FL((FL))
    FJ --> FM((FM))
    FK -- X --> FN((FN))
    FL -- X --> FO((FO))
    FM --> FP((FP))
    FN -- X --> FQ((FQ))
    FO -- X --> FR((FR))
    FP --> FS((FS))
    FQ -- X --> FT((FT))
    FR -- X --> FU((FU))
    FS --> FV((FV))
    FT -- X --> FW((FW))
    FU -- X --> FX((FX))
    FV --> FY((FY))
    FW -- X --> FZ((FZ))
    FX -- X --> GA((GA))
    FY -- X --> GB((GB))
    FZ --> GC((GC))
    GA --> GD((GD))
    GB -- X --> GE((GE))
    GC --> GF((GF))
    GD --> GG((GG))
    GE -- X --> GH((GH))
    GF --> GI((GI))
    GG --> GJ((GJ))
    GH -- X --> GK((GK))
    GI -- X --> GL((GL))
    GJ --> GM((GM))
    GK -- X --> GN((GN))
    GL -- X --> GO((GO))
    GM --> GP((GP))
    GN -- X --> GQ((GQ))
    GO -- X --> GR((GR))
    GP --> GS((GS))
    GQ -- X --> GT((GT))
    GR -- X --> GU((GU))
    GS --> GV((GV))
    GT -- X --> GW((GW))
    GU -- X --> GX((GX))
    GV --> GY((GY))
    GW -- X --> GZ((GZ))
    GX -- X --> HA((HA))
    GY -- X --> HB((HB))
    GZ --> HC((HC))
    HA --> HD((HD))
    HB -- X --> HE((HE))
    HC --> HF((HF))
    HD --> HG((HG))
    HE -- X --> HH((HH))
    HF --> HI((HI))
    HG --> HJ((HJ))
    HH -- X --> HK((HK))
    HI -- X --> HL((HL))
    HJ --> HM((HM))
    HK -- X --> HN((HN))
    HL -- X --> HO((HO))
    HM --> HP((HP))
    HN -- X --> HQ((HQ))
    HO -- X --> HR((HR))
    HP --> HS((HS))
    HQ -- X --> HT((HT))
    HR -- X --> HU((HU))
    HS --> HV((HV))
    HT -- X --> HW((HW))
    HU -- X --> HX((HX))
    HV --> HY((HY))
    HW -- X --> HZ((HZ))
    HX -- X --> IA((IA))
    HY -- X --> IB((IB))
    HZ --> IC((IC))
    IA --> ID((ID))
    IB -- X --> IE((IE))
    IC --> IF((IF))
    ID --> IG((IG))
    IE -- X --> IH((IH))
    IF --> II((II))
    IG --> IJ((IJ))
    IH -- X --> IK((IK))
    II -- X --> IL((IL))
    IJ --> IM((IM))
    IK -- X --> IN((IN))
    IL -- X --> IO((IO))
    IM --> IP((IP))
    IN -- X --> IQ((IQ))
    IO -- X --> IR((IR))
    IP --> IS((IS))
    IQ -- X --> IT((IT))
    IR -- X --> IU((IU))
    IS --> IV((IV))
    IT -- X --> IW((IW))
    IU -- X --> IX((IX))
    IV --> IY((IY))
    IW -- X --> IZ((IZ))
    IX -- X --> JA((JA))
    IY -- X --> JB((JB))
    IZ --> JC((JC))
    JA --> JD((JD))
    JB -- X --> JE((JE))
    JC --> JF((JF))
    JD --> JG((JG))
    JE -- X --> JH((JH))
    JF --> JI((JI))
    JG --> JJ((JJ))
    JH -- X --> JK((JK))
    JI -- X --> JL((JL))
    JJ --> JM((JM))
    JK -- X --> JN((JN))
    JL -- X --> JO((JO))
    JM --> JP((JP))
    JN -- X --> JQ((JQ))
    JO -- X --> JR((JR))
    JP --> JS((JS))
    JQ -- X --> JT((JT))
    JR -- X --> JU((JU))
    JS --> JV((JV))
    JT -- X --> JW((JW))
    JU -- X --> JX((JX))
    JV --> JY((JY))
    JW -- X --> JZ((JZ))
    JX -- X --> KA((KA))
    JY -- X --> KB((KB))
    JZ --> KC((KC))
    KA --> KD((KD))
    KB -- X --> KE((KE))
    KC --> KF((KF))
    KD --> KG((KG))
    KE -- X --> KH((KH))
    KF --> KI((KI))
    KG --> KJ((KJ))
    KH -- X --> KK((KK))
    KI -- X --> KL((KL))
    KJ --> KM((KM))
    KK -- X --> KN((KN))
    KL -- X --> KO((KO))
    KM --> KP((KP))
    KN -- X --> KQ((KQ))
    KO -- X --> KR((KR))
    KP --> KS((KS))
    KQ -- X --> KT((KT))
    KR -- X --> KU((KU))
    KS --> KV((KV))
    KT -- X --> KW((KW))
    KU -- X --> KX((KX))
    KV --> KY((KY))
    KW -- X --> KZ((KZ))
    KX -- X --> LA((LA))
    KY -- X --> LB((LB))
    KZ --> LC((LC))
    LA --> LD((LD))
    LB -- X --> LE((LE))
    LC --> LF((LF))
    LD --> LG((LG))
    LE -- X --> LH((LH))
    LF --> LI((LI))
    LG --> LJ((LJ))
    LH -- X --> LK((LK))
    LI -- X --> LL((LL))
    LJ --> LM((LM))
    LK -- X --> LN((LN))
    LL -- X --> LO((LO))
    LM --> LP((LP))
    LN -- X --> LQ((LQ))
    LO -- X --> LR((LR))
    LP --> LS((LS))
    LQ -- X --> LT((LT))
    LR -- X --> LU((LU))
    LS --> LV((LV))
    LT -- X --> LW((LW))
    LU -- X --> LX((LX))
    LV --> LY((LY))
    LW -- X --> LZ((LZ))
    LX -- X --> MA((MA))
    LY -- X --> MB((MB))
    LZ --> MC((MC))
    MA --> MD((MD))
    MB -- X --> ME((ME))
    MC --> MF((MF))
    MD --> MG((MG))
    ME -- X --> MH((MH))
    MF --> MI((MI))
    MG --> MJ((MJ))
    MH -- X --> MK((MK))
    MI -- X --> ML((ML))
    MJ --> MM((MM))
    MK -- X --> MN((MN))
    ML -- X --> MO((MO))
    MM --> MP((MP))
    MN -- X --> MQ((MQ))
    MO -- X --> MR((MR))
    MP --> MS((MS))
    MQ -- X --> MT((MT))
    MR -- X --> MU((MU))
    MS --> MV((MV))
    MT -- X --> MW((MW))
    MU -- X --> MX((MX))
    MV --> MY((MY))
    MW -- X --> MZ((MZ))
    MX -- X --> NA((NA))
    MY -- X --> NB((NB))
    MZ --> NC((NC))
    NA --> ND((ND))
    NB -- X --> NE((NE))
    NC --> NF((NF))
    ND --> NG((NG))
    NE -- X --> NH((NH))
    NF --> NI((NI))
    NG --> NJ((NJ))
    NH -- X --> NK((NK))
    NI -- X --> NL((NL))
    NJ --> NM((NM))
    NK -- X --> NN((NN))
    NL -- X --> NO((NO))
    NM --> NP((NP))
    NN -- X --> NQ((NQ))
    NO -- X --> NR((NR))
    NP --> NS((NS))
    NQ -- X --> NT((NT))
    NR -- X --> NU((NU))
    NS --> NV((NV))
    NT -- X --> NW((NW))
    NU -- X --> NX((NX))
    NV --> NY((NY))
    NW -- X --> NZ((NZ))
    NX -- X --> OA((OA))
    NY -- X --> OB((OB))
    NZ --> OC((OC))
    OA --> OD((OD))
    OB -- X --> OE((OE))
    OC --> OF((OF))
    OD --> OG((OG))
    OE -- X --> OH((OH))
    OF --> OI((OI))
    OG --> OJ((OJ))
    OH -- X --> OK((OK))
    OI -- X --> OL((OL))
    OJ --> OM((OM))
    OK -- X --> ON((ON))
    OL -- X --> OO((OO))
    OM --> OP((OP))
    ON -- X --> OQ((OQ))
    OO -- X --> OR((OR))
    OP --> OS((OS))
    OQ -- X --> OT((OT))
    OR -- X --> OU((OU))
    OS --> OV((OV))
    OT -- X --> OW((OW))
    OU -- X --> OX((OX))
    OV --> OY((OY))
    OW -- X --> OZ((OZ))
    OX -- X --> PA((PA))
    OY -- X --> PB((PB))
    OZ --> PC((PC))
    PA --> PD((PD))
    PB -- X --> PE((PE))
    PC --> PF((PF))
    PD --> PG((PG))
    PE -- X --> PH((PH))
    PF --> PI((PI))
    PG --> PJ((PJ))
    PH -- X --> PK((PK))
    PI -- X --> PL((PL))
    PJ --> PM((PM))
    PK -- X --> PN((PN))
    PL -- X --> PO((PO))
    PM --> PP((PP))
    PN -- X --> PQ((PQ))
    PO -- X --> PR((PR))
    PP --> PS((PS))
    PQ -- X --> PT((PT))
    PR -- X --> PU((PU))
    PS --> PV((PV))
    PT -- X --> PW((PW))
    PU -- X --> PX((PX))
    PV --> PY((PY))
    PW -- X --> PZ((PZ))
    PX -- X --> QA((QA))
    PY -- X --> QB((QB))
    PZ --> QC((QC))
    QA --> QD((QD))
    QB -- X --> QE((QE))
    QC --> QF((QF))
    QD --> QG((QG))
    QE -- X --> QH((QH))
    QF --> QI((QI))
    QG --> QJ((QJ))
    QH -- X --> QK((QK))
    QI -- X --> QL((QL))
    QJ --> QM((QM))
    QK -- X --> QN((QN))
    QL -- X --> QO((QO))
    QM --> QP((QP))
    QN -- X --> QQ((QQ))
    QO -- X --> QR((QR))
    QP --> QS((QS))
    QQ -- X --> QT((QT))
    QR -- X --> QU((QU))
    QS --> QV((QV))
    QT -- X --> QW((QW))
    QU -- X --> QX((QX))
    QV --> QY((QY))
    QW -- X --> QZ((QZ))
    QX -- X --> RA((RA))
    QY -- X --> RB((RB))
    QZ --> RC((RC))
    RA --> RD((RD))
    RB -- X --> RE((RE))
    RC --> RF((RF))
    RD --> RG((RG))
    RE -- X --> RH((RH))
    RF --> RI((RI))
    RG --> RJ((RJ))
    RH -- X --> RK((RK))
    RI -- X --> RL((RL))
    RJ --> RM((RM))
    RK -- X --> RN((RN))
    RL -- X --> RO((RO))
    RM --> RP((RP))
    RN -- X --> RQ((RQ))
    RO -- X --> RR((RR))
    RP --> RS((RS))
    RQ -- X --> RT((RT))
    RR -- X --> RU((RU))
    RS --> RV((RV))
    RT -- X --> RW((RW))
    RU -- X --> RX((RX))
    RV --> RY((RY))
    RW -- X --> RZ((RZ))
    RX -- X --> SA((SA))
    RY -- X --> SB((SB))
    RZ --> SC((SC))
    SA --> SD((SD))
    SB -- X --> SE((SE))
    SC --> SF((SF))
    SD --> SG((SG))
    SE -- X --> SH((SH))
    SF --> SI((SI))
    SG --> SJ((SJ))
    SH -- X --> SK((SK))
    SI -- X --> SL((SL))
    SJ --> SM((SM))
    SK -- X --> SN((SN))
    SL -- X --> SO((SO))
    SM --> SP((SP))
    SN -- X --> SQ((SQ))
    SO -- X --> SR((SR))
    SP --> SS((SS))
    SQ -- X --> ST((ST))
    SR -- X --> SU((SU))
    SS --> SV((SV))
    ST -- X --> SW((SW))
    SU -- X --> SX((SX))
    SV --> SY((SY))
    SW -- X --> SZ((SZ))
    SX -- X --> TA((TA))
    SY -- X --> TB((TB))
    SZ --> TC((TC))
    TA --> TD((TD))
    TB -- X --> TE((TE))
    TC --> TF((TF))
    TD --> TG((TG))
    TE -- X --> TH((TH))
    TF --> TI((TI))
    TG --> TJ((TJ))
    TH -- X --> TK((TK))
    TI -- X --> TL((TL))
    TJ --> TM((TM))
    TK -- X --> TN((TN))
    TL -- X --> TO((TO))
    TM --> TP((TP))
    TN -- X --> TQ((TQ))
    TO -- X --> TR((TR))
    TP --> TS((TS))
    TQ -- X --> TT((TT))
    TR -- X --> TU((TU))
    TS --> TV((TV))
    TT -- X --> TW((TW))
    TU -- X --> TX((TX))
    TV --> TY((TY))
    TW -- X --> TZ((TZ))
    TX -- X --> UA((UA))
    TY -- X --> UB((UB))
    TZ --> UC((UC))
    UA --> UD((UD))
    UB -- X --> UE((UE))
    UC --> UF((UF))
    UD --> UG((UG))
    UE -- X --> UH((UH))
    UF --> UI((UI))
    UG --> UJ((UJ))
    UH -- X --> UK((UK))
    UI -- X --> UL((UL))
    UJ --> UM((UM))
    UK -- X --> UN((UN))
    UL -- X --> UO((UO))
    UM --> UP((UP))
    UN -- X --> UQ((UQ))
    UO -- X --> UR((UR))
    UP --> US((US))
    UQ -- X --> UT((UT))
    UR -- X --> UJ2((UJ2))
    US --> UV((UV))
    UT -- X --> UW((UW))
    UJ2 -- X --> UX((UX))
    UV --> UY((UY))
    UW -- X --> UZ((UZ))
    UX -- X --> VA((VA))
    UY -- X --> VB((VB))
    UZ --> VC((VC))
    VA --> VD((VD))
    VB -- X --> VE((VE))
    VC --> VF((VF))
    VD --> VG((VG))
    VE -- X --> VH((VH))
    VF --> VI((VI))
    VG --> VJ((VJ))
    VH -- X --> VK((VK))
    VI -- X --> VL((VL))
    VJ --> VM((VM))
    VK -- X --> VN((VN))
    VL -- X --> VO((VO))
    VM --> VP((VP))
    VN -- X --> VQ((VQ))
    VO -- X --> VR((VR))
    VP --> VS((VS))
    VQ -- X --> VT((VT))
    VR -- X --> VJ3((VJ3))
    VS --> VV((VV))
    VT -- X --> VW((VW))
    VJ3 -- X --> VX((VX))
    VV --> VY((VY))
    VW -- X --> VZ((VZ))
    VX -- X --> WA((WA))
    VY -- X --> WB((WB))
    VZ --> WC2((WC2))
    WA --> WD2((WD2))
    WB -- X --> WE2((WE2))
    WC2 --> WF2((WF2))
    WD2 --> WG2((WG2))
    WE2 -- X --> WH2((WH2))
    WF2 --> WI2((WI2))
    WG2 --> WJ2((WJ2))
    WH2 -- X --> WK2((WK2))
    WI2 -- X --> WL2((WL2))
    WJ2 --> WM2((WM2))
    WK2 -- X --> WN2((WN2))
    WL2 -- X --> WO2((WO2))
    WM2 --> WP2((WP2))
    WN2 -- X --> WQ2((WQ2))
    WO2 -- X --> WR2((WR2))
    WP2 --> WS2((WS2))
    WQ2 -- X --> WT2((WT2))
    WR2 -- X --> WJ4((WJ4))
    WS2 --> WV2((WV2))
    WT2 -- X --> WW2((WW2))
    WJ4 -- X --> WX2((WX2))
    WV2 --> WY2((WY2))
    WW2 -- X --> WZ2((WZ2))
    WX2 -- X --> XA((XA))
    WY2 -- X --> XB((XB))
    WZ2 --> XC2((XC2))
    XA --> XD2((XD2))
    XB -- X --> XE2((XE2))
    XC2 --> XF2((XF2))
    XD2 --> XG2((XG2))
    XE2 -- X --> XH2((XH2))
    XF2 --> XI2((XI2))
    XG2 --> XJ2((XJ2))
    XH2 -- X --> XK2((XK2))
    XI2 -- X --> XL2((XL2))
    XJ2 --> XM2((XM2))
    XK2 -- X --> XN2((XN2))
    XL2 -- X --> XO2((XO2))
    XM2 --> XP2((XP2))
    XN2 -- X --> XQ2((XQ2))
    XO2 -- X --> XR2((XR2))
    XP2 --> XS2((XS2))
    XQ2 -- X --> XT2((XT2))
    XR2 -- X --> XJ5((XJ5))
    XS2 --> XV2((XV2))
    XT2 -- X --> XW2((XW2))
    XJ5 -- X --> XX2((XX2))
    XV2 --> XY2((XY2))
    XW2 -- X --> XZ2((XZ2))
    XX2 -- X --> YA((YA))
    XY2 -- X --> YB((YB))
    XZ2 --> YC2((YC2))
    YA --> YD2((YD2))
    YB -- X --> YE2((YE2))
    YC2 --> YF2((YF2))
    YD2 --> YG2((YG2))
    YE2 -- X --> YH2((YH2))
    YF2 --> YI2((YI2))
    YG2 --> YJ2((YJ2))
    YH2 -- X --> YK2((YK2))
    YI2 -- X --> YL2((YL2))
    YJ2 --> YM2((YM2))
    YK2 -- X --> YN2((YN2))
    YL2 -- X --> YO2((YO2))
    YM2 --> YP2((YP2))
    YN2 -- X --> YQ2((YQ2))
    YO2 -- X --> YR2((YR2))
    YP2 --> YS2((YS2))
    YQ2 -- X --> YT2((YT2))
    YR2 -- X --> YJ6((YJ6))
    YS2 --> YV2((YV2))
    YT2 -- X --> YW2((YW2))
    YJ6 -- X --> YX2((YX2))
    YV2 --> YY2((YY2))
    YW2 -- X --> YZ2((YZ2))
    YX2 -- X --> ZA((ZA))
    YY2 -- X --> ZB((ZB))
    YZ2 --> ZC2((ZC2))
    ZA --> ZD2((ZD2))
    ZB -- X --> ZE2((ZE2))
    ZC2 --> ZF2((ZF2))
    ZD2 --> ZG2((ZG2))
    ZE2 -- X --> ZH2((ZH2))
    ZF2 --> ZI2((ZI2))
    ZG2 --> ZJ2((ZJ2))
    ZH2 -- X --> ZK2((ZK2))
    ZI2 -- X --> ZL2((ZL2))
    ZJ2 --> ZM2((ZM2))
    ZK2 -- X --> ZN2((ZN2))
    ZL2 -- X --> ZO2((ZO2))
    ZM2 --> ZP2((ZP2))
    ZN2 -- X --> ZQ2((ZQ2))
    ZO2 -- X --> ZR2((ZR2))
    ZP2 --> ZS2((ZS2))
    ZQ2 -- X --> ZT2((ZT2))
    ZR2 -- X --> ZJ7((ZJ7))
    ZS2 --> ZV2((ZV2))
    ZT2 -- X --> ZW2((ZW2))
    ZJ7 -- X --> ZX2((ZX2))
    ZV2 --> ZY2((ZY2))
    ZW2 -- X --> ZZ2((ZZ2))
    ZX2 -- X --> AA2((AA2))
    ZY2 -- X --> AB2((AB2))
    ZZ2 --> AC2((AC2))
    AA2 --> AD2((AD2))
    AB2 -- X --> AE2((AE2))
    AC2 --> AF2((AF2))
    AD2 --> AG2((AG2))
    AE2 -- X --> AH2((AH2))
    AF2 --> AI2((AI2))
    AG2 --> AJ2((AJ2))
    AH2 -- X --> AK2((AK2))
    AI2 -- X --> AL2((AL2))
    AJ2 --> AM2((AM2))
    AK2 -- X --> AN2((AN2))
    AL2 -- X --> AO2((AO2))
    AM2 --> AP2((AP2))
    AN2 -- X --> AQ2((AQ2))
    AO2 -- X --> AR2((AR2))
    AP2 --> AS2((AS2))
    AQ2 -- X --> AT2((AT2))
    AR2 -- X --> AU2((AU2))
    AS2 --> AV2((AV2))
    AT2 -- X --> AW2((AW2))
    AU2 -- X --> AX2((AX2))
    AV2 --> AY2((AY2))
    AW2 -- X --> AZ2((AZ2))
    AX2 -- X --> BA2((BA2))
    AY2 -- X --> BB2((BB2))
    AZ2 --> BC2((BC2))
    BA2 --> BD2((BD2))
    BB2 -- X --> BE2((BE2))
    BC2 --> BF2((BF2))
    BD2 --> BG2((BG2))
    BE2 -- X --> BH2((BH2))
    BF2 --> BI2((BI2))
    BG2 --> BJ2((BJ2))
    BH2 -- X --> BK2((BK2))
    BI2 -- X --> BL2((BL2))
    BJ2 --> BM2((BM2))
    BK2 -- X --> BN2((BN2))
    BL2 -- X --> BO2((BO2))
    BM2 --> BP2((BP2))
    BN2 -- X --> BQ2((BQ2))
    BO2 -- X --> BR2((BR2))
    BP2 --> BS2((BS2))
    BQ2 -- X --> BT2((BT2))
    BR2 -- X --> BJ8((BJ8))
    BS2 --> BV2((BV2))
    BT2 -- X --> BW2((BW2))
    BJ8 -- X --> BX2((BX2))
    BV2 --> BY2((BY2))
    BW2 -- X --> BZ2((BZ2))
    BX2 -- X --> CA2((CA2))
    BY2 -- X --> CB2((CB2))
    BZ2 --> CC2((CC2))
    CA2 --> CD2((CD2))
    CB2 -- X --> CE2((CE2))
    CC2 --> CF2((CF2))
    CD2 --> CG2((CG2))
    CE2 -- X --> CH2((CH2))
    CF2 --> CI2((CI2))
    CG2 --> CJ2((CJ2))
    CH2 -- X --> CK2((CK2))
    CI2 -- X --> CL2((CL2))
    CJ2 --> CM2((CM2))
    CK2 -- X --> CN2((CN2))
    CL2 -- X --> CO2((CO2))
    CM2 --> CP2((CP2))
    CN2 -- X --> CQ2((CQ2))
    CO2 -- X --> CR2((CR2))
    CP2 --> CS2((CS2))
    CQ2 -- X --> CT2((CT2))
    CR2 -- X --> CJ9((CJ9))
    CS2 --> CV2((CV2))
    CT2 -- X --> CW2((CW2))
    CJ9 -- X --> CX2((CX2))
    CV2 --> CY2((CY2))
    CW2 -- X --> CZ2((CZ2))
    CX2 -- X --> DA2((DA2))
    CY2 -- X --> DB2((DB2))
    CZ2 --> DC2((DC2))
    DA2 --> DD2((DD2))
    DB2 -- X --> DE2((DE2))
    DC2 --> DF2((DF2))
    DD2 --> DG2((DG2))
    DE2 -- X --> DH2((DH2))
    DF2 --> DI2((DI2))
    DG2 --> DJ2((DJ2))
    DH2 -- X --> DK2((DK2))
    DI2 -- X --> DL2((DL2))
    DJ2 --> DM2((DM2))
    DK2 -- X --> DN2((DN2))
    DL2 -- X --> DO2((DO2))
    DM2 --> DP2((DP2))
    DN2 -- X --> DQ2((DQ2))
    DO2 -- X --> DR2((DR2))
    DP2 --> DS2((DS2))
    DQ2 -- X --> DT2((DT2))
    DR2 -- X --> DJ10((DJ10))
    DS2 --> DV2((DV2))
    DT2 -- X --> DW2((DW2))
    DJ10 -- X --> DX2((DX2))
    DV2 --> DY2((DY2))
    DW2 -- X --> DZ2((DZ2))
    DX2 -- X --> EA2((EA2))
    DY2 -- X --> EB2((EB2))
    DZ2 --> EC2((EC2))
    EA2 --> ED2((ED2))
    EB2 -- X --> EE2((EE2))
    EC2 --> EF2((EF2))
    ED2 --> EG2((EG2))
    EE2 -- X --> EH2((EH2))
    EF2 --> EI2((EI2))
    EG2 --> EJ2((EJ2))
    EH2 -- X --> EK2((EK2))
    EI2 -- X --> EL2((EL2))
    EJ2 --> EM2((EM2))
    EK2 -- X --> EN2((EN2))
    EL2 -- X --> EO2((EO2))
    EM2 --> EP2((EP2))
    EN2 -- X --> EQ2((EQ2))
    EO2 -- X --> ER2((ER2))
    EP2 --> ES2((ES2))
    EQ2 -- X --> ET2((ET2))
    ER2 -- X --> DJ11((DJ11))
    ES2 --> EV2((EV2))
    ET2 -- X --> EW2((EW2))
    DJ11 -- X --> EX2((EX2))
    EV2 --> EY2((EY2))
    EW2 -- X --> EZ2((EZ2))
    EX2 -- X --> FA2((FA2))
    EY2 -- X --> FB2((FB2))
    EZ2 --> FC2((FC2))
    FA2 --> FD2((FD2))
    FB2 -- X --> FE2((FE2))
    FC2 --> FF2((FF2))
    FD2 --> FG2((FG2))
    FE2 -- X --> FH2((FH2))
    FF2 --> FI2((FI2))
    FG2 --> FJ2((FJ2))
    FH2 -- X --> FK2((FK2))
    FI2 -- X --> FL2((FL2))
    FJ2 --> FM2((FM2))
    FK2 -- X --> FN2((FN2))
    FL2 -- X --> FO2((FO2))
    FM2 --> FP2((FP2))
    FN2 -- X --> FQ2((FQ2))
    FO2 -- X --> FR2((FR2))
    FP2 --> FS2((FS2))
    FQ2 -- X --> FT2((FT2))
    FR2 -- X --> DJ12((DJ12))
    FS2 --> FV2((FV2))
    FT2 -- X --> FW2((FW2))
    DJ12 -- X --> FX2((FX2))
    FV2 --> FY2((FY2))
    FW2 -- X --> FZ2((FZ2))
    FX2 -- X --> GA2((GA2))
    FY2 -- X --> GB2((GB2))
    FZ2 --> GC2((GC2))
    GA2 --> GD2((GD2))
    GB2 -- X --> GE2((GE2))
    GC2 --> GF2((GF2))

```



Multiprocess Architecture – Chrome Browser

- Many web browsers ran as single process (some still do)
 - If one web site causes trouble, entire browser can hang or crash
- Google Chrome Browser is multiprocess with 3 different types of processes:
 - **Browser** process manages user interface, disk and network I/O
 - **Renderer** process renders web pages, deals with HTML, Javascript. A new renderer created for each website opened
 - ▶ Runs in **sandbox** restricting disk and network I/O, minimizing effect of security exploits
 - **Plug-in** process for each type of plug-in





Interprocess Communication (IPC)

- Processes within a system may be **independent** or **cooperating**
- Cooperating process can affect or be affected by other processes, including sharing data
- Reasons for cooperating processes:
 - Information sharing
 - Computation speedup
 - Modularity
 - Convenience
- Cooperating processes need **interprocess communication (IPC)**
- Two models of IPC

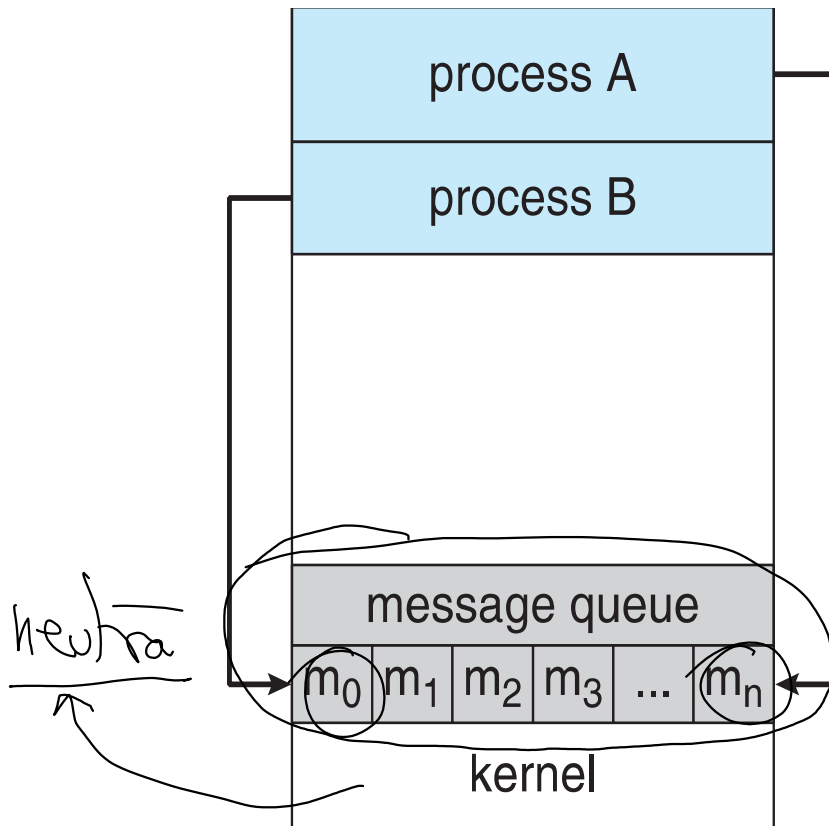
- LARS
- **Shared memory**
 - **Message passing**



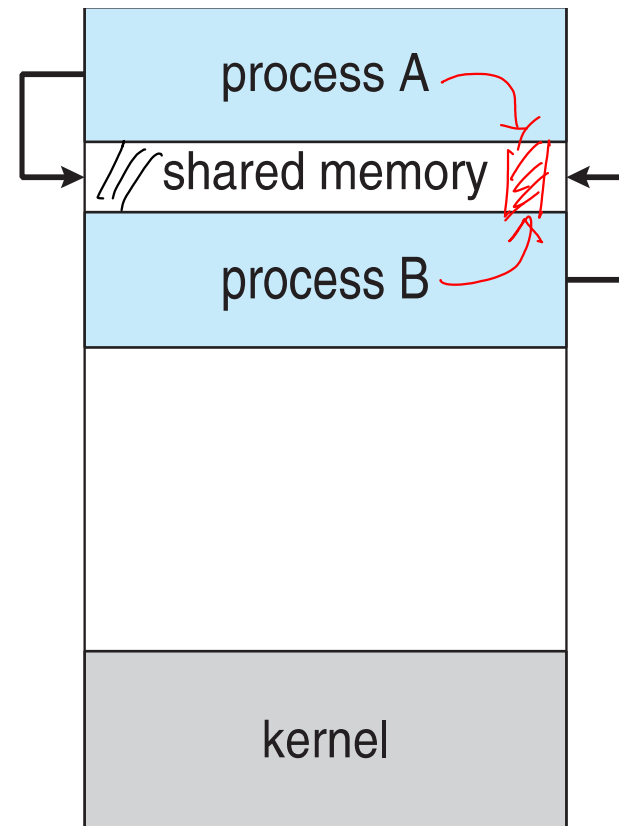


Communications Models

(a) Message passing. (b) shared memory.



(a)



(b)





Cooperating Processes

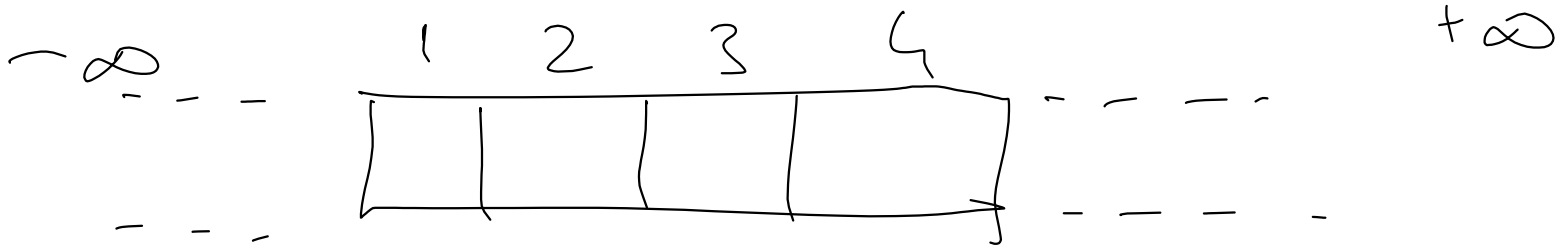
- **Independent** process cannot affect or be affected by the execution of another process
- **Cooperating** process can affect or be affected by the execution of another process
- Advantages of process cooperation
 - Information sharing
 - Computation speed-up
 - Modularity
 - Convenience





Producer-Consumer Problem

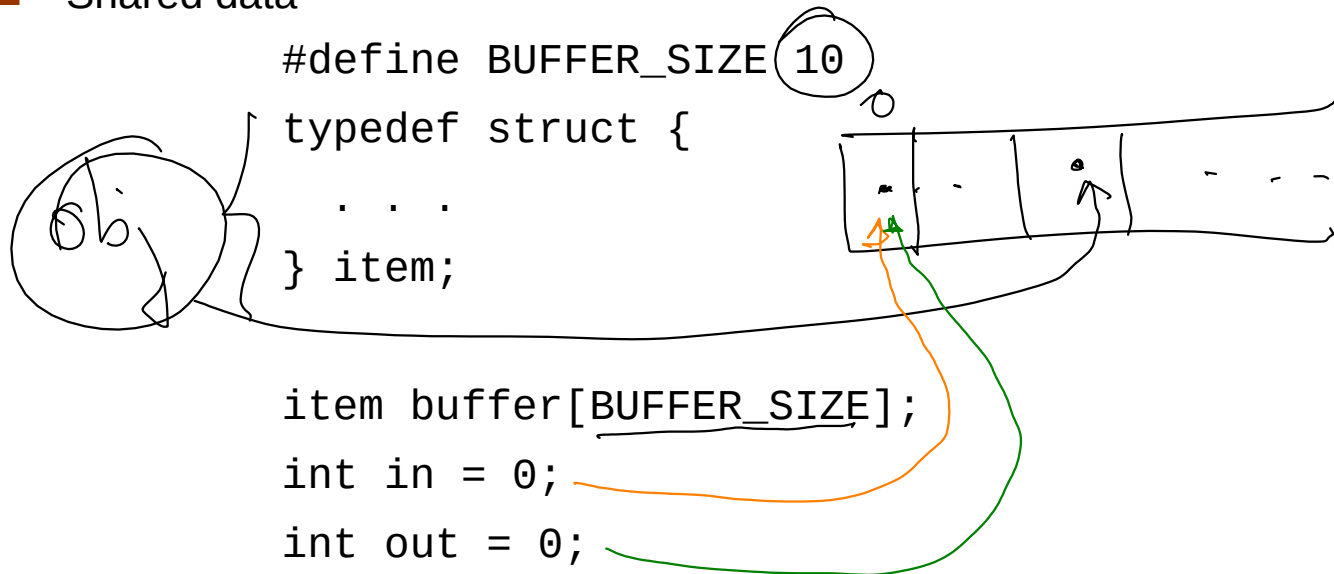
- Paradigm for cooperating processes, *producer* process produces information that is consumed by a *consumer* process
 - **unbounded-buffer** places no practical limit on the size of the buffer
 - **bounded-buffer** assumes that there is a fixed buffer size





Bounded-Buffer – Shared-Memory Solution

- Shared data



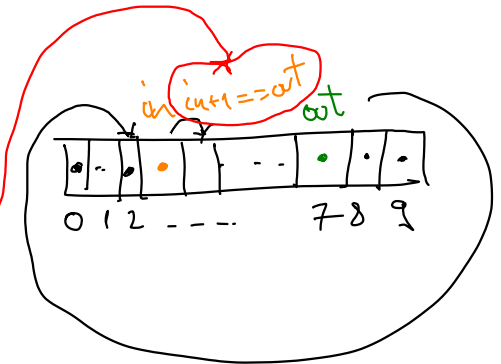
- Solution is correct, but can only use $BUFFER_SIZE - 1$ elements





Bounded-Buffer – Producer

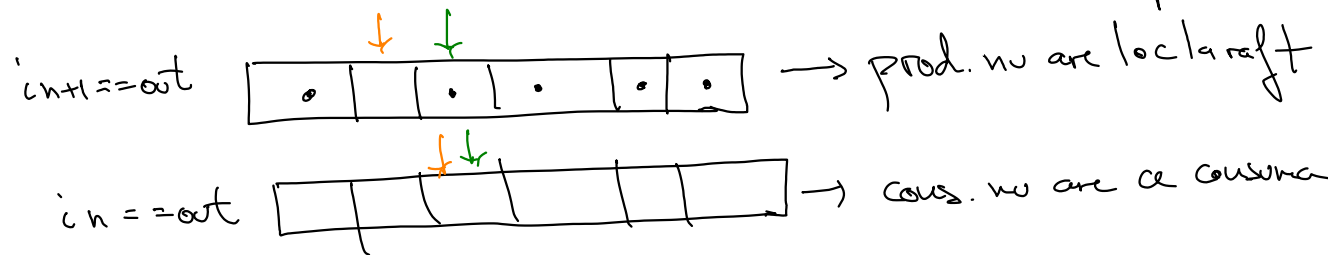
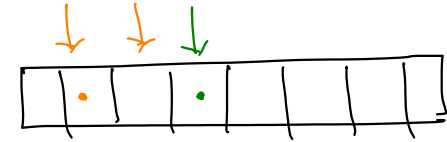
```
item next_produced;  
→ while (true) {  
    → /* produce an item in next produced */  
    → while (((in + 1) % BUFFER_SIZE) == out)  
        ; /* do nothing */ block  
    buffer[in] = next_produced;  
    in = (in + 1) % BUFFER_SIZE;  
}
```





Bounded Buffer – Consumer

```
item next_consumed;  
→ while (true) {  
    while (in == out)  
        ; /* do nothing */  
    next_consumed = buffer[out];  
    out = (out + 1) % BUFFER_SIZE;  
    → /* consume the item in next consumed */  
}
```





Interprocess Communication – Shared Memory

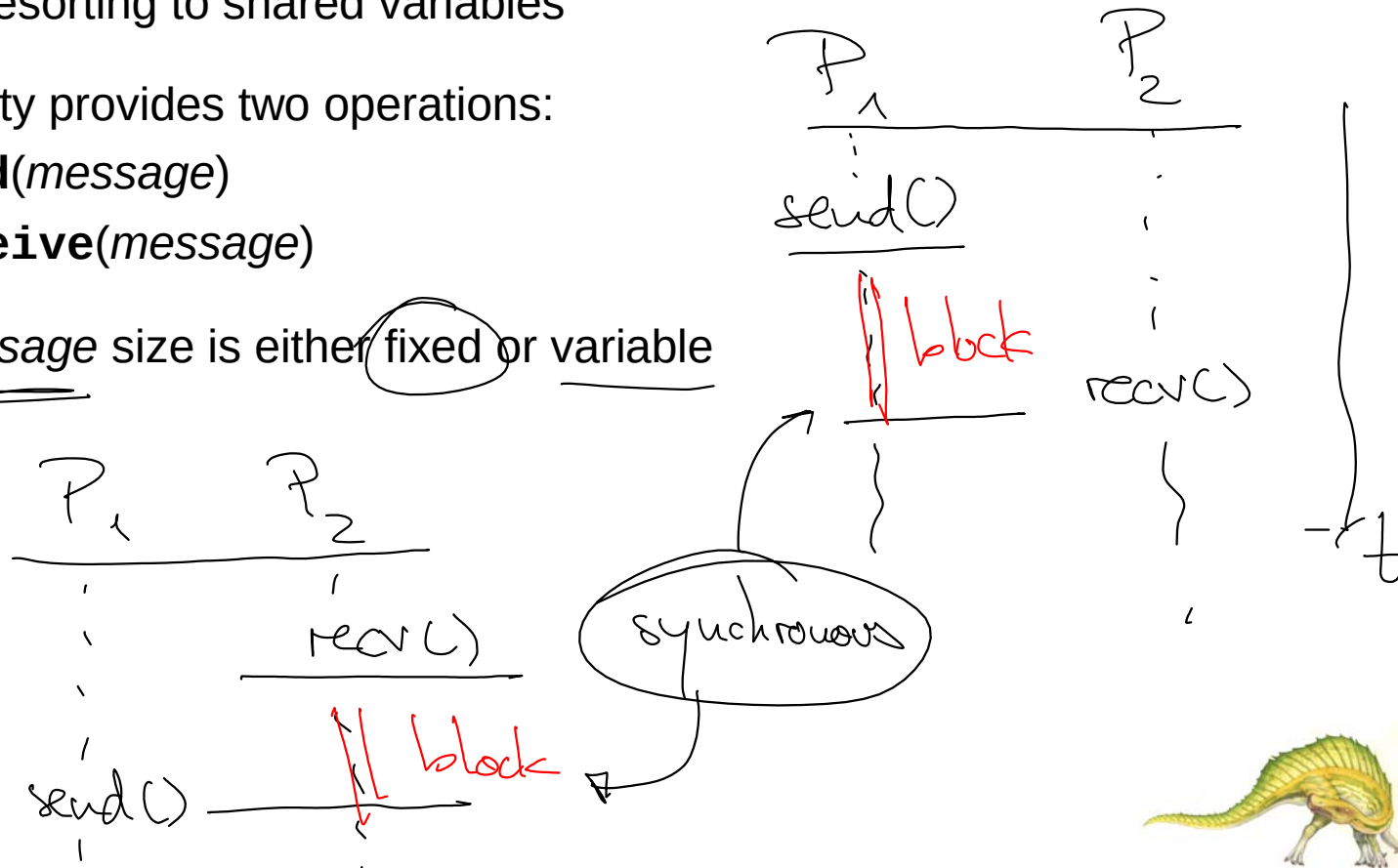
- An area of memory shared among the processes that wish to communicate
- ■ The communication is under the control of the users processes not the operating system.
- Major issues is to provide mechanism that will allow the user processes to synchronize their actions when they access shared memory.
- ■ Synchronization is discussed in great details in Chapter 5.





Interprocess Communication – Message Passing

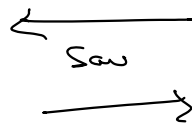
- Mechanism for processes to communicate and to synchronize their actions
- Message system – processes communicate with each other without resorting to shared variables
- IPC facility provides two operations:
 - **send(message)**
 - **receive(message)**
- The message size is either fixed or variable





Message Passing (Cont.)

- If processes P and Q wish to communicate, they need to:
 - Establish a ***communication link*** between them
 - Exchange messages via send/receive
- Implementation issues:
 - How are links established?
 - Can a link be associated with more than two processes?
 - How many links can there be between every pair of communicating processes?
 - What is the capacity of a link? (message size e.g.)
 - Is the size of a message that the link can accommodate fixed or variable?
- Is a link unidirectional or bi-directional?



full-duplex
half-duplex





Message Passing (Cont.)

■ Implementation of communication link

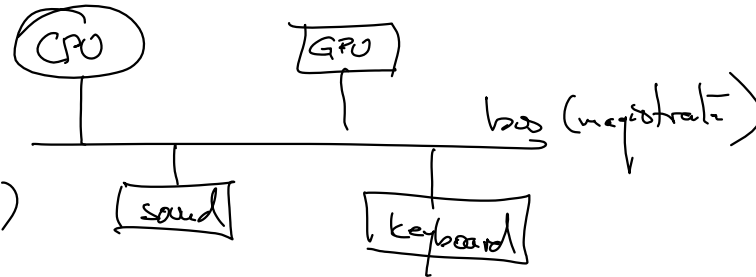
● Physical:

- ▶ Shared memory
- ▶ Hardware bus

① Network (MPI e.g.)

● Logical:

- ▶ Direct or indirect
- ▶ Synchronous or asynchronous
- ▶ Automatic or explicit buffering





Direct Communication

- Processes must name each other explicitly:
 - **send** (P , *message*) – send a message to process P
 - **receive**(Q , *message*) – receive a message from process Q
- Properties of communication link
 - Links are established automatically
 - A link is associated with exactly one pair of communicating processes
 - Between each pair there exists exactly one link
 - The link may be unidirectional, but is usually bi-directional





Indirect Communication

- Messages are directed and received from mailboxes (also referred to as ports)
 - Each mailbox has a unique id
 - Processes can communicate only if they share a mailbox
- Properties of communication link
 - Link established only if processes share a common mailbox
 - A link may be associated with many processes
 - Each pair of processes may share several communication links
 - Link may be unidirectional or bi-directional





Indirect Communication

■ Operations

- create a new mailbox (port)
- send and receive messages through mailbox
- destroy a mailbox

■ Primitives are defined as:

send(A, message) – send a message to mailbox A

receive(A, message) – receive a message from mailbox A

↓
fin. mailbox no : 80 plain
: 443 crypto

my server no : 8080
: 5555





Indirect Communication

■ Mailbox sharing

- P_1 , P_2 , and P_3 share mailbox A
- P_1 sends; P_2 and P_3 receive

○ Who gets the message?

■ Solutions

- ● Allow a link to be associated with at most two processes
- ● Allow only one process at a time to execute a receive operation
- ● Allow the system to select arbitrarily the receiver.
Sender is notified who the receiver was.

• broadcast





Synchronization

- Message passing may be either blocking or non-blocking
- **Blocking** is considered **synchronous**
 - **Blocking send** -- the sender is blocked until the message is received
 - **Blocking receive** -- the receiver is blocked until a message is available
- **Non-blocking** is considered **asynchronous**
 - **Non-blocking send** -- the sender sends the message and continue
 - **Non-blocking receive** -- the receiver receives:
 - A valid message, or
 - **Null** message
- Different combinations possible
 - If both send and receive are blocking, we have a **rendezvous**





Synchronization (Cont.)

- Producer-consumer becomes trivial

```
message next_produced;  
→ while (true) {  
    /* produce an item in next produced */  
    send(next_produced);  
}  
  
message next_consumed;  
while (true) {  
    receive(next_consumed);  
  
    /* consume the item in next consumed */  
}
```





Buffering

- Queue of messages attached to the link.
- implemented in one of three ways
 1. Zero capacity – no messages are queued on a link.
Sender must wait for receiver (rendezvous)
 2. Bounded capacity – finite length of n messages
Sender must wait if link full
 3. Unbounded capacity – infinite length
Sender never waits





Examples of IPC Systems - POSIX

■ POSIX Shared Memory

- Process first creates shared memory segment

```
shm_fd = shm_open(name, 0 CREAT | 0 RDWR, 0666);
```

ugo

- Also used to open an existing segment to share it

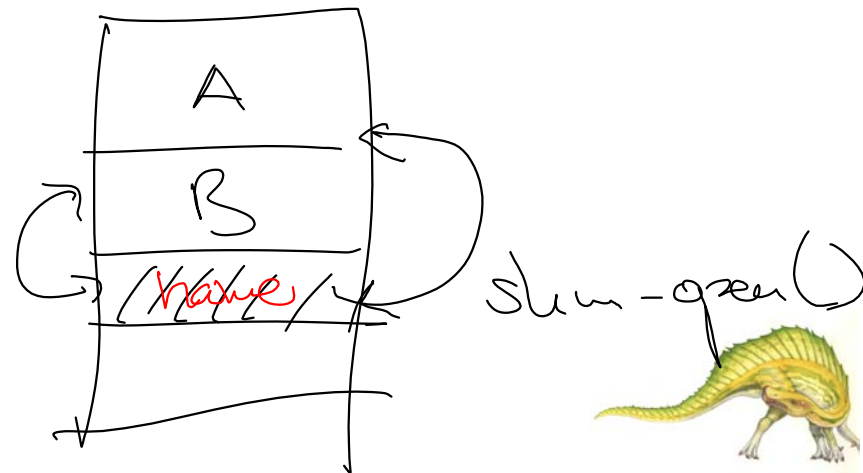
- Set the size of the object

```
ftruncate(shm fd, 4096);
```

molt
✓ 4KB on Linux

- Now the process could write to the shared memory

```
sprintf(shared memory, "Writing to shared memory");
```





IPC POSIX Producer

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <sys/shm.h>
#include <sys/stat.h>

int main()
{
    /* the size (in bytes) of shared memory object */
    const int SIZE = 4096; 4k
    /* name of the shared memory object */
    const char *name = "OS";
    /* strings written to shared memory */
    const char *message_0 = "Hello";
    const char *message_1 = "World!"; { => shm

    /* shared memory file descriptor */
    int shm_fd;
    /* pointer to shared memory object */
    void *ptr;

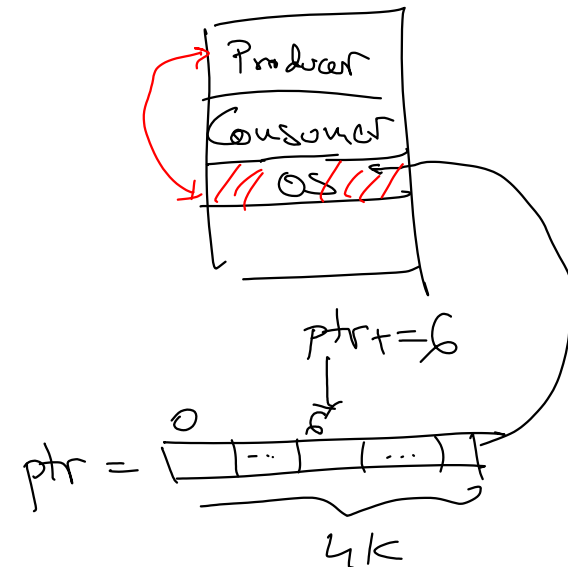
    /* create the shared memory object */
    shm_fd = shm_open(name, O_CREAT | O_RDWR, 0666);

    /* configure the size of the shared memory object */
    ftruncate(shm_fd, SIZE);
    4k

    /* memory map the shared memory object */
    ptr = mmap(0, SIZE, PROT_WRITE, MAP_SHARED, shm_fd, 0);
    4k

    /* write to the shared memory object */
    sprintf(ptr, "%s", message_0);
    ptr += strlen(message_0);
    sprintf(ptr, "%s", message_1);
    ptr += strlen(message_1);

    return 0;
}
```





IPC POSIX Consumer

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/shm.h>
#include <sys/stat.h>

int main()
{
    /* the size (in bytes) of shared memory object */
    const int SIZE = 4096;
    /* name of the shared memory object */
    const char *name = "OS";
    /* shared memory file descriptor */
    → int shm_fd;
    /* pointer to shared memory object */
    → void *ptr;

    /* open the shared memory object */
    shm_fd = shm_open(name, O_RDONLY, 0666);

    /* memory map the shared memory object */
    ptr = mmap(0, SIZE, PROT_READ, MAP_SHARED, shm_fd, 0);

    /* read from the shared memory object */
    printf("%s", (char *)ptr);

    /* remove the shared memory object */
    → shm_unlink(name);

    return 0;
}
```





Examples of IPC Systems - Mach

- Mach communication is message based
 - Even system calls are messages
 - Each task gets two mailboxes at creation- Kernel and Notify
 - Only three system calls needed for message transfer
msg_send(), **msg_receive()**, **msg_rpc()**
 - Mailboxes needed for communication, created via
port_allocate()
 - Send and receive are flexible, for example four options if mailbox full:
 - ▶ Wait indefinitely
 - ▶ Wait at most n milliseconds
 - ▶ Return immediately
 - ▶ Temporarily cache a message





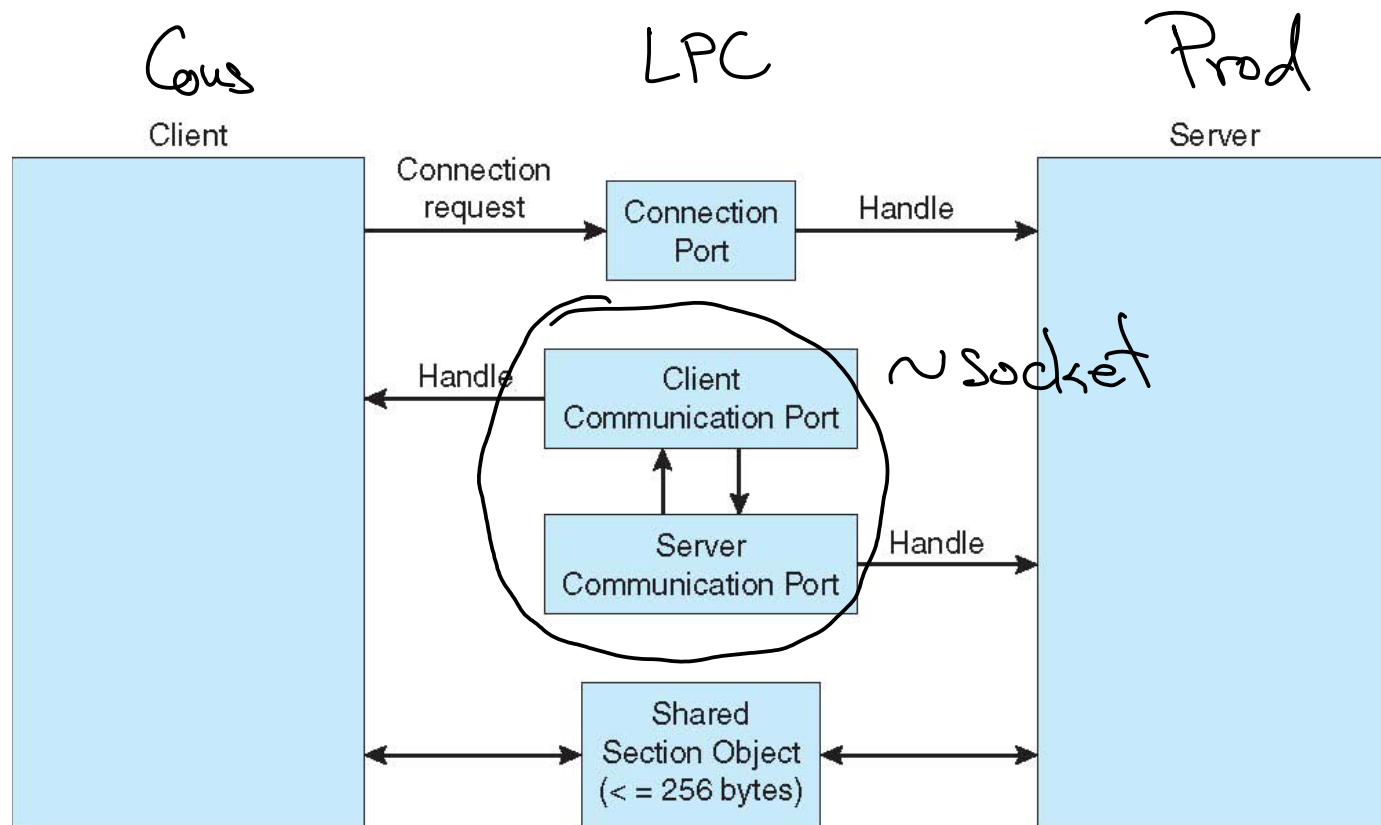
Examples of IPC Systems – Windows

- Message-passing centric via **advanced local procedure call (LPC)** facility ~ RPC (Remote Procedure Call)
 - Only works between processes on the same system
 - Uses ports (like mailboxes) to establish and maintain communication channels
 - Communication works as follows:
 - ▶ The client opens a handle to the subsystem's **connection port** object.
 - ▶ The client sends a connection request.
 - ▶ The server creates two private **communication ports** and returns the handle to one of them to the client.
 - ▶ The client and server use the corresponding port handle to send messages or callbacks and to listen for replies.





Local Procedure Calls in Windows





Communications in Client-Server Systems

- Sockets
- Remote Procedure Calls
- Pipes
- Remote Method Invocation (Java)





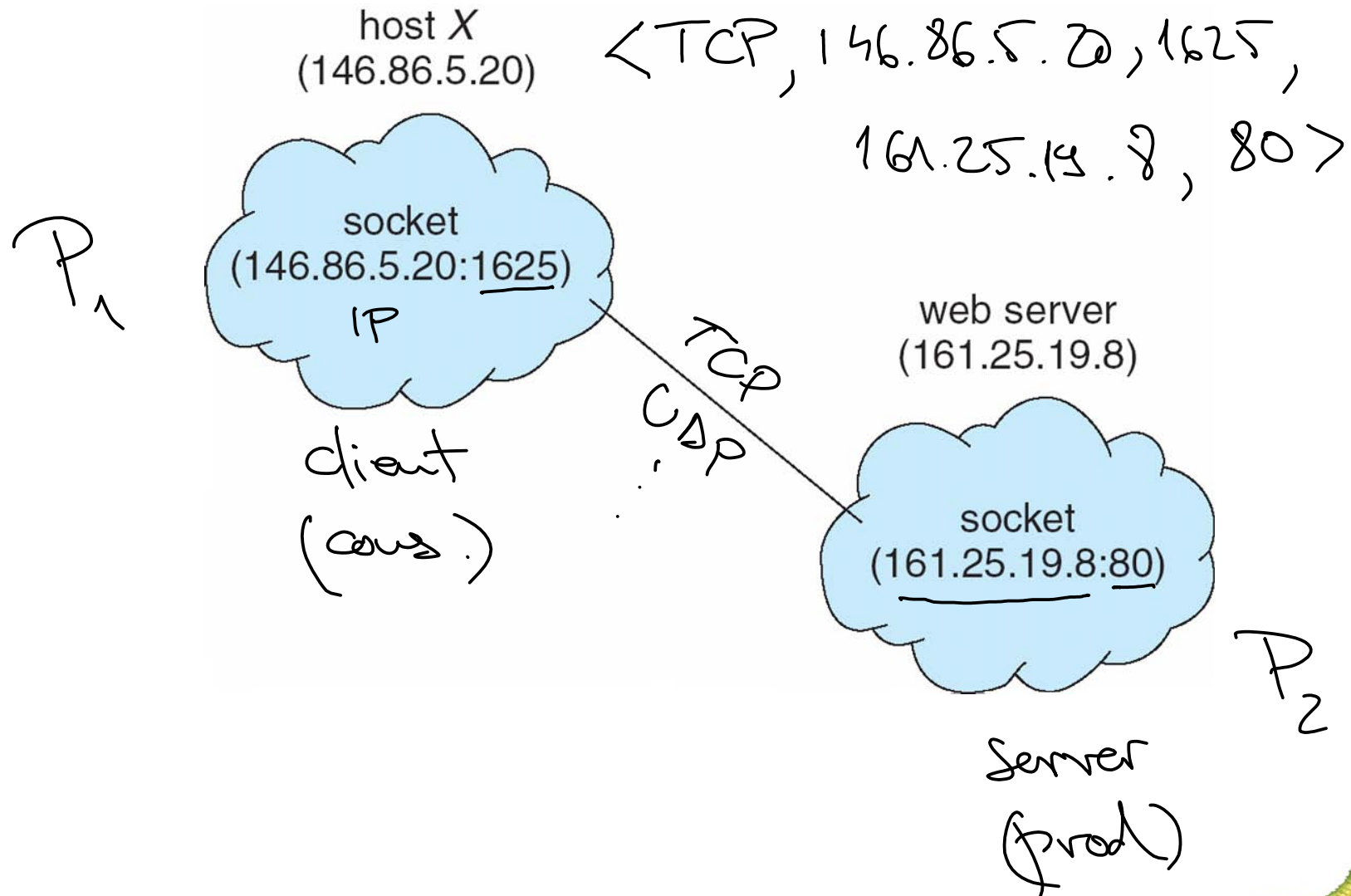
Sockets

- A **socket** is defined as an endpoint for communication
- Concatenation of IP address and **port** – a number included at start of message packet to differentiate network services on a host
- The socket **161.25.19.8:1625** refers to port **1625** on host **161.25.19.8**
- Communication consists between a pair of sockets
- All ports below 1024 are **well known**, used for standard services
- Special IP address 127.0.0.1 (**loopback**) to refer to system on which process is running





Socket Communication





Sockets in Java

- Three types of sockets
 - **Connection-oriented (TCP)**
 - **Connectionless (UDP)**
 - **MulticastSocket** class— data can be sent to multiple recipients
- Consider this “Date” server:

```
import java.net.*;
import java.io.*;

public class DateServer
{
    public static void main(String[] args) {
        try {
            ServerSocket sock = new ServerSocket(6013);

            /* now listen for connections */
            while (true) {
                Socket client = sock.accept();

                PrintWriter pout = new
                    PrintWriter(client.getOutputStream(), true);

                /* write the Date to the socket */
                pout.println(new java.util.Date().toString());

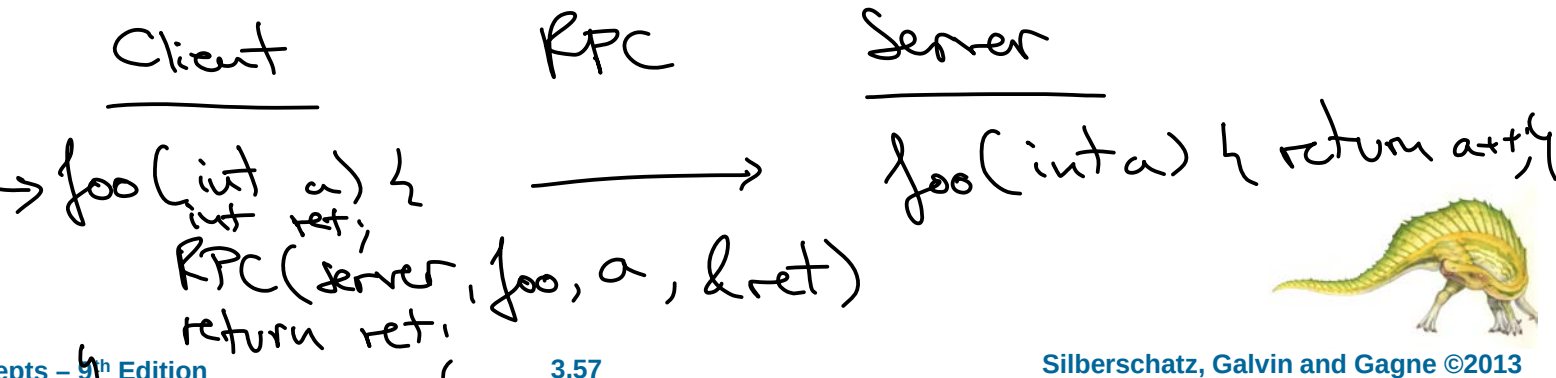
                /* close the socket and resume */
                /* listening for connections */
                client.close();
            }
        }
        catch (IOException ioe) {
            System.err.println(ioe);
        }
    }
}
```





Remote Procedure Calls

- Remote procedure call (RPC) abstracts procedure calls between processes on networked systems
 - Again uses ports for service differentiation
- **Stubs** – client-side proxy for the actual procedure on the server
- The client-side stub locates the server and marshalls the parameters
- The server-side stub receives this message, unpacks the marshalled parameters, and performs the procedure on the server
- On Windows, stub code compile from specification written in **Microsoft Interface Definition Language (MIDL)**





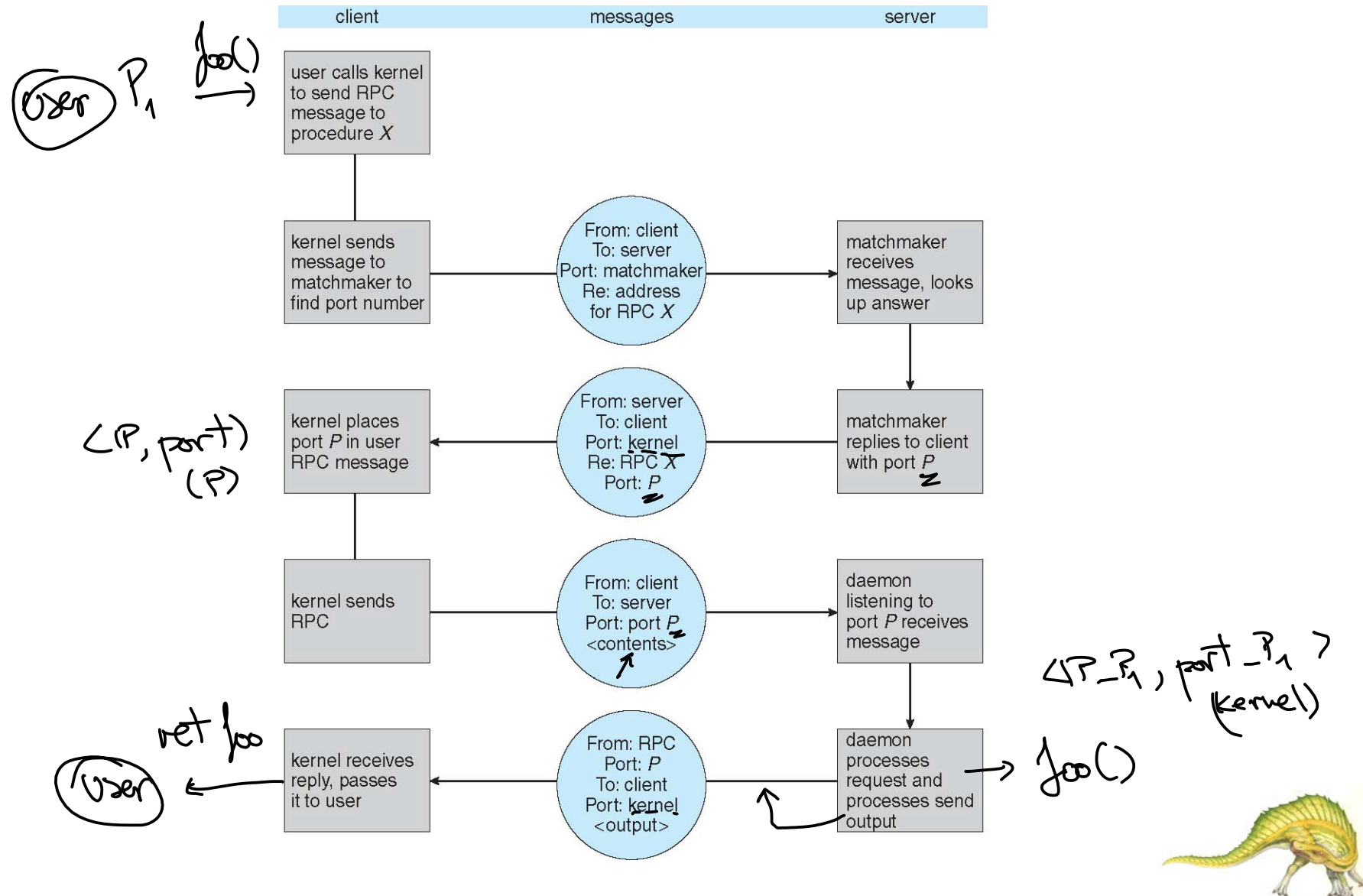
Remote Procedure Calls (Cont.)

- Data representation handled via **External Data Representation (XDL)** format to account for different architectures
 - **Big-endian** and **little-endian**
- Remote communication has more failure scenarios than local
 - Messages can be delivered ***exactly once*** rather than ***at most once***
- OS typically provides a rendezvous (or **matchmaker**) service to connect client and server





Execution of RPC





Pipes

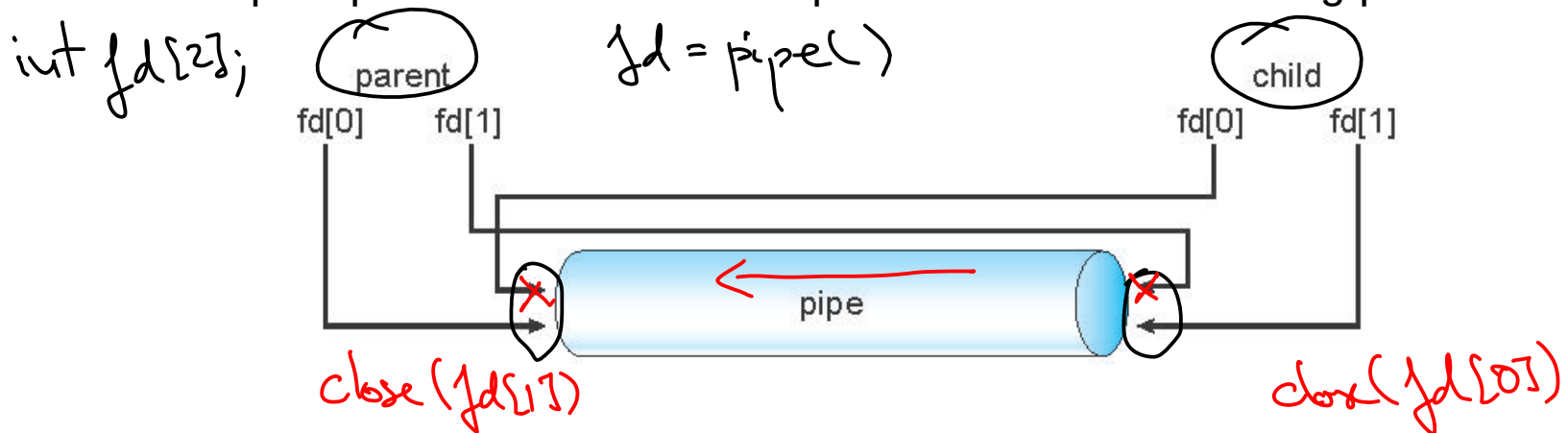
- Acts as a conduit allowing two processes to communicate
- Issues:
 - Is communication unidirectional or bidirectional?
 - In the case of two-way communication, is it half or full-duplex?
 - Must there exist a relationship (i.e., ***parent-child***) between the communicating processes?
 - Can the pipes be used over a network?
- Ordinary pipes – cannot be accessed from outside the process that created it. Typically, a parent process creates a pipe and uses it to communicate with a child process that it created.
- Named pipes – can be accessed without a parent-child relationship.





Ordinary Pipes

- Ordinary Pipes allow communication in standard producer-consumer style
- Producer writes to one end (the **write-end** of the pipe)
- Consumer reads from the other end (the **read-end** of the pipe)
- Ordinary pipes are therefore unidirectional
- Require parent-child relationship between communicating processes



- Windows calls these **anonymous pipes**
- See Unix and Windows code samples in textbook





Named Pipes

- Named Pipes are more powerful than ordinary pipes
- Communication is bidirectional
- No parent-child relationship is necessary between the communicating processes
- Several processes can use the named pipe for communication
- Provided on both UNIX and Windows systems



End of Chapter 3

