

Unit 14: The Mach Operating System

14.1. Mach Overview and System Concepts

The Mach Operating System

- Research project at Carnegie Mellon University (CMU)
- based on a simple communication-oriented kernel
- designed to support distributed and parallel computation
- provides UNIX 4.3BSD compatibility
- small, extensible system kernel provides:
 - Processor scheduling
 - Management of virtual memory
 - Interprocess communication

History of Mach

- RIG (Rochester Intelligent Gateway)
 - Research OS for 16-bit Data General minicomputer (Eclipse)
 - University of Rochester, Richard Rashid et al., 1975
 - Demonstrate modular structuring of an OS
 - Message-passing communication
- Accent
 - Message-passing OS at Carnegie Mellon University (Rashid moved to CMU in 1979)
 - PERQ machine with bitmapped screen, mouse, network
 - Protection, transparent networking and 32-bit virtual memory manag.
 - Operational by 1981
 - Accent was running on 150 PERQs by 1984

History of Mach (contd.)

- Mach
 - Third-generation OS
 - Compatible with UNIX
 - Many improvements over Accent
 - threads, better interprocess communication, multiprocessor support, novel virtual memory system
- DARPA Strategic Computing Initiative
 - U.S. Department of Defense's Advanced Research Projects Agency was searching for a multiprocessor operating system
 - CMU was selected, substantial DARPA funding
 - Mach was combined with 4.2/4.3BSD code
 - Large kernel but absolute compatibility with Berkeley UNIX (DARPA requirement)

History of Mach (contd.)

- Mach 2 was released
 - VAX 11/784 version (4 CPU multiprocessor) in 1986
 - Ports to IBM PC/RT and Sun3 in 1987
 - Versions for Encore and Sequent multiprocessors available in 1987
- OSF/1
 - Open Software Foundation (consortium of vendors; IBM, DEC, HP,...) selected Mach 2.5 as basis for OSF/1
 - Alliance against AT&T and Sun Microsystem's UNIX System V.4
 - Large and monolithic kernel
- Mach 3 microkernel OS
 - CMU removed all BSD code from kernel and put it in user space
 - User-level OS emulator for BSD UNIX in 1988.

Mach Components

- Kernel exports a small number of abstractions through an integrated interface.
- Operating system support environments provide:
 - distributed file access
 - transparent network interprocess communication
 - remote execution facilities
 - UNIX 4.3BSD emulation
- Many traditional operating system functions can be implemented by user programs or servers outside the kernel.

Mach provides features not found in UNIX 4.3BSD:

- Multiple tasks, each with a large, paged virtual memory space
- Multiple threads of execution within each task, with a flexible scheduling facility
- Flexible sharing of memory between tasks
- Efficient and consistent message-based interprocess communication
- Memory-mapped files
- Transparent network extensibility
- A flexible, capability-based approach to security and protection
- Support for multiprocessor scheduling

Tasks and Threads

- Mach splits the notion of a process into two abstractions, the task and the thread:
- **Task**
 - environment for program execution
 - basic unit of protection
 - basic unit of resource allocation, including
 - paged virtual address space
 - port rights that protect access to system resources
 - The task itself performs no computation; it is a framework for running threads.

Tasks and Threads (contd.)

- **Thread**

- basic unit of execution
- lightweight process executing within a task – defined by processor state
- executes within the context of a single task
- each task may contain more than one thread
- All threads within a task share
 - the virtual memory address space and
 - communication rights of that task.
- basic unit of scheduling
- multiple threads from one task may be executing simultaneously.

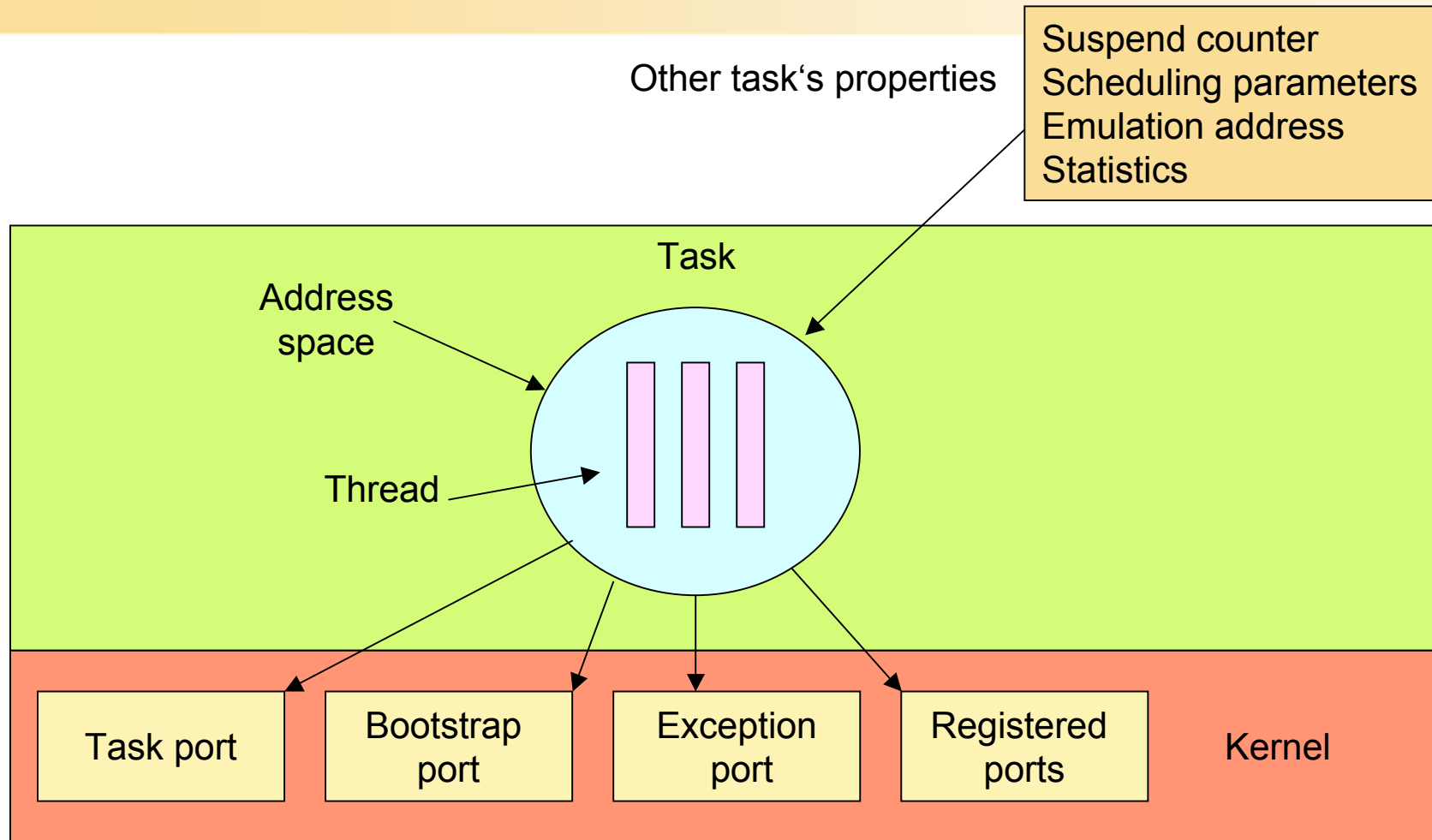
Task and Thread Ports

- Tasks and threads are represented by ports (message queues).
- Task port and thread port tell the kernel which task or thread is executing a function call.
 - `task_self()` and `thread_self()` return task and thread ports of the currently executing thread.
- Tasks can have access to the task and thread ports of other tasks and threads.
 - creator gets access to a new task port or thread port
 - any thread can pass access to ports in a message to other threads

Task and Thread Ports (contd.)

- Access rights to a task or thread port allow
 - to act on behalf of that task or thread
 - to perform Mach function calls
- Access to a task's port indirectly permits access to all threads within that task.
- The task port and thread port are often called kernel ports.
- Tasks and threads have a number of special ports associated with them.
 - Notify port, Exception port, Bootstrap port.

A Mach Task



Mach Ports and Messages

- Communication among operating system objects is achieved through messages.
- Mach messaging is implemented by three kernel abstractions:
- **Port**
 - protected communication channel
 - implemented as a finite-length message queue)
 - basic object reference mechanism in Mach
 - similar to that of object references in an object-oriented system
- Operations are requested by sending messages to and from the ports that represent objects.
- When a task is created, a port that represents the task is simultaneously created.
- When the task is destroyed, its port is also destroyed.

Mach Ports and Messages (contd.)

- **Port set**
 - group of ports, combining the message queues of the constituent ports.
 - may be used to receive a message sent to any of several ports.
- **Messages**
 - Used to communicate between objects;
 - data stream consisting of two parts:
 - fixed-length header
 - variable-length message body -- typed data objects
 - header contains information about:
 - size of the message, its type, and its destination
 - body contains the content (or a pointer to the content) of the message
 - Messages may be of any size, may contain:
 - in-line data, pointers to data, and capabilities for ports.
- A single message may transfer the entire address space of a task.

Port Access Rights

- Communication between objects is protected by a system of port access rights (Capabilities).
- Send access to a port
 - Implies that a message can be sent to that port.
- Receive access to a port
 - Allows a message to be de-queued from that port.
 - Only one task may have receive access for a given port at a time;
 - more than one thread within that task may concurrently attempt to receive messages
 - receive access implies send rights.
- Multiple tasks may hold send rights to the same port, but
 - only one task at a time may hold receive rights to a port.
- Port access rights can be passed in messages.

Port Sets

- Port sets are
 - a bag holding zero or more receive rights.
 - a mechanism to allow a thread to block while waiting for a message sent to any of several ports.
- A port may be a member of no more than one port set at any time, and a task can have only one port set.
 - `port_set_allocate()`, `port_set_add()`,
 - `port_set_remove()`, `port_set_status()`, `port_set_deallocate()`.
- Unlike port rights, a port set right can't be passed in messages.

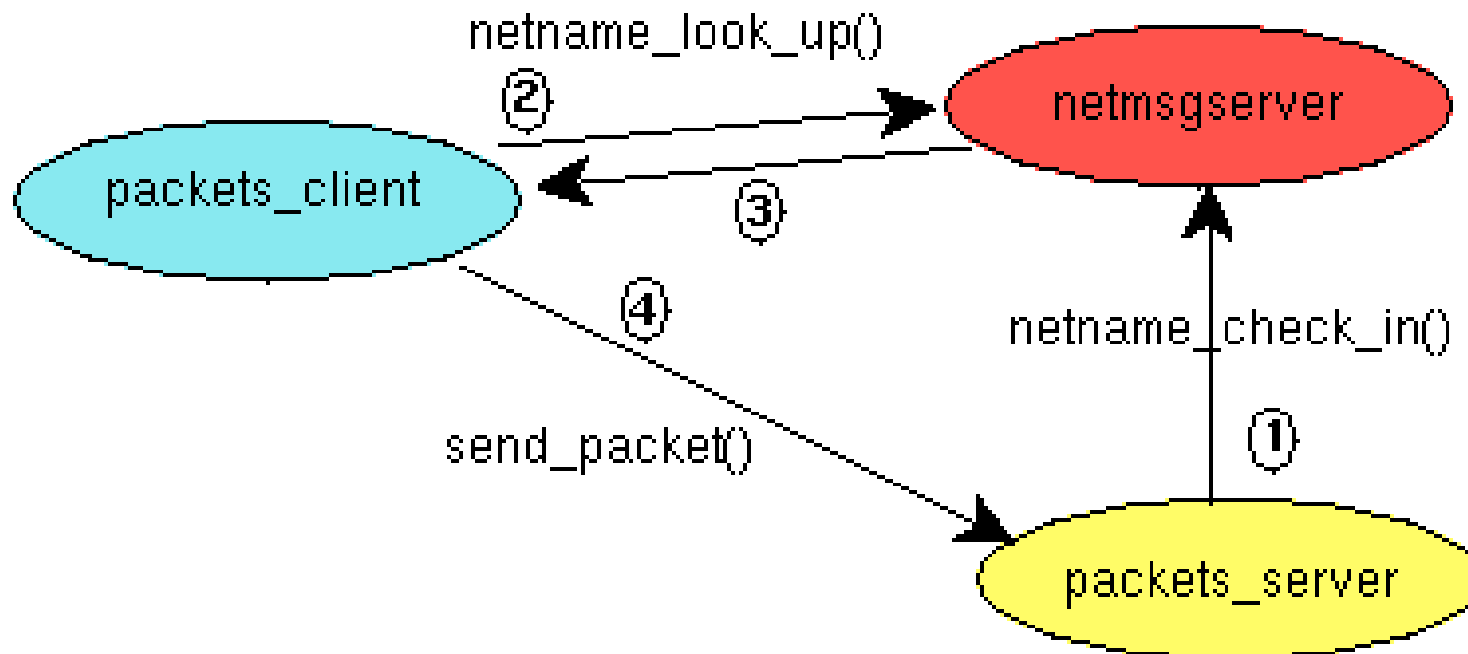
Port Names

- Every task has its own port name space, used for port and port set names.
 - For example, one task with receive rights for a port may know the port by the name 13,
 - while another task with send rights for the same port may know it by the name 17.
- A task has only one name for a port.
- Typically, these names are small integers, but this is implementation dependent.
 - `port_rename()` call can be used to change a task's name for a port.

Port Queues

- Messages that are sent to a port are held there until removed by a thread.
 - The queue associated with a port is of finite length and may become full.
- A thread sending to a filled queue has a choice of three alternatives:
 - By default, the sender is suspended until it can successfully transmit the message.
 - The sender can have the kernel hold the message for later transmission.
 - If the sender selects this action, it can't transmit further messages to the port until the kernel notifies it that the port has received the initial message.
- The attempt to send a message to a full port can be reported to the sender as an error.

Client/Server Setup



Programming with Ports (IPC)

- **Allocating a port**

```
# include <mach/mach.h>
# include <mach/port.h>

int allocate_port(port_name_t* port) {
    /* allocate a new port */
    kern_return_t ret;
    ret = port_allocate( task_self(), port );
    if (ret != KERN_SUCCESS) {
        mach_error("port_allocate:", ret );
        return -1;
    }
    return 0;
}
```

Registering with the Name Service

```
# include <mach/mach.h>
# include <servers/netname.h>
# include <mach/message.h>
# include <mach/port.h>

int allocate_and_check_in_port(char* name, port_name_t* port) {
/* allocate a new port and check the name in with netmsgserver*/
    kern_return_t ret;
    netname_name_t n_name;
    ret = port_allocate( task_self(), port );
    if (ret != NETNAME_SUCCESS) return -1;
    strncpy(n_name, name, sizeof(n_name));
    ret = netname_check_in( name_server_port, n_name,
                           task_self(), *port );
    if (ret != NETNAME_SUCCESS) return -2;
    return 0;
}
```

Looking up a Port

```
int lookup_port( char * name, port_name_t * port ) {
    /* lookup a port registered with netmsgserver */
    kern_return_t ret;
    netname_name_t n_name;

    strncpy(n_name, name, sizeof(n_name));
    /*the pseudo name "*" initiates a broadcast on the net*/
    if ((ret = netname_lookup(name_server_port, "*",
                            n_name, port)) != NETNAME_SUCCESS) {
        mach_error("netname_lookup", ret);
        return -1;
    }
    return 0;
}
```

Programming with Mach IPC

- Interface definition file packets.defs - Mach 3

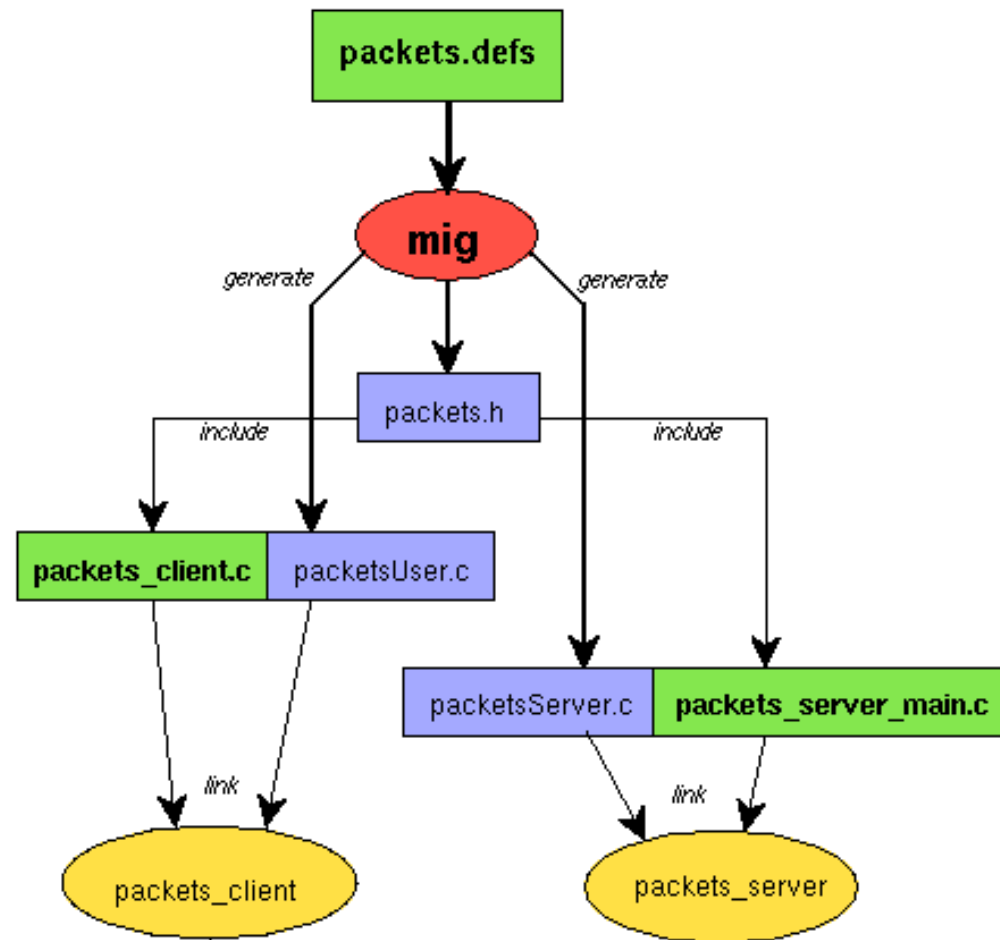
```
subsystem packets 0;
ServerPrefix Serv_;

# include <mach/mach_types.defs>

routine send_packet(
    RequestPort server: mach_port_t;
    in          count:      int
);

simpleroutine server_exit(
    RequestPort server: mach_port_t
);
```

Mach Interface Generator (MIG)



MIG Operations

	Nachricht senden	Antwort empfangen	Fehler- meldung zurück	Rück- gabe- wert
<code>simpleroutine</code>	✓		✓	
<code>routine</code>	✓	✓	✓	
<code>simpleprocedure</code>	✓			
<code>procedure</code>	✓	✓		
<code>function</code>	✓	✓		✓

MIG Declarations - EBNF

operation	::=	operation-type op-name (parameter-list); function op-name (parameter-list) : func- type;
operation-type	::=	simpleroutine routine simpleprocedure procedure.
parameter-list	::=	parameter { ; parameter }.
parameter	::=	[specification] var-name : type [, dealloc- flag].
specification	::=	in out inout RequestPort ReplyPort WaitTime MsgType.

Network-transparent IPC

- netmsgserver extends reach of local IPC
 - uses TCP/IP to transmit IPC messages to remote sites
 - provides a network-wide name service for port lookup

