PART 2

Organization Of An Operating System

Services An OS Supplies

- Support for concurrent execution
- Facilities for process synchronization
- Inter-process communication mechanisms
- Facilities for message passing and asynchronous events
- Management of address spaces and virtual memory
- Protection among users and running applications
- High-level interface for I/O devices
- A file system and file access facilities
- Network communication

Operating System From The Inside

- Well-understood subsystems
- Many subsystems employ heuristic policies
  - Policies can conflict
  - Heuristics can have corner cases
- Complexity arises from interactions among subsystems
- Side-effects can be
  - Unintended
  - Unanticipated

Building An Operating System

- The intellectual challenge comes from the “system”, not from individual pieces
- Structured design is needed
- It can be difficult to understand the consequences of choices
- We will use a hierarchical microkernel design to help control complexity

Major OS Components

- Process Manager
- Memory Manager
- Device Manger
- Clock (time) Manager
- File Manager
- Interprocess Communication
- Intermachine Communication
- Accounting

Multilevel Structure

- The design paradigm we will use
- Organizes components
- Controls interactions among subsystems
- Allows a system to be understood and built incrementally

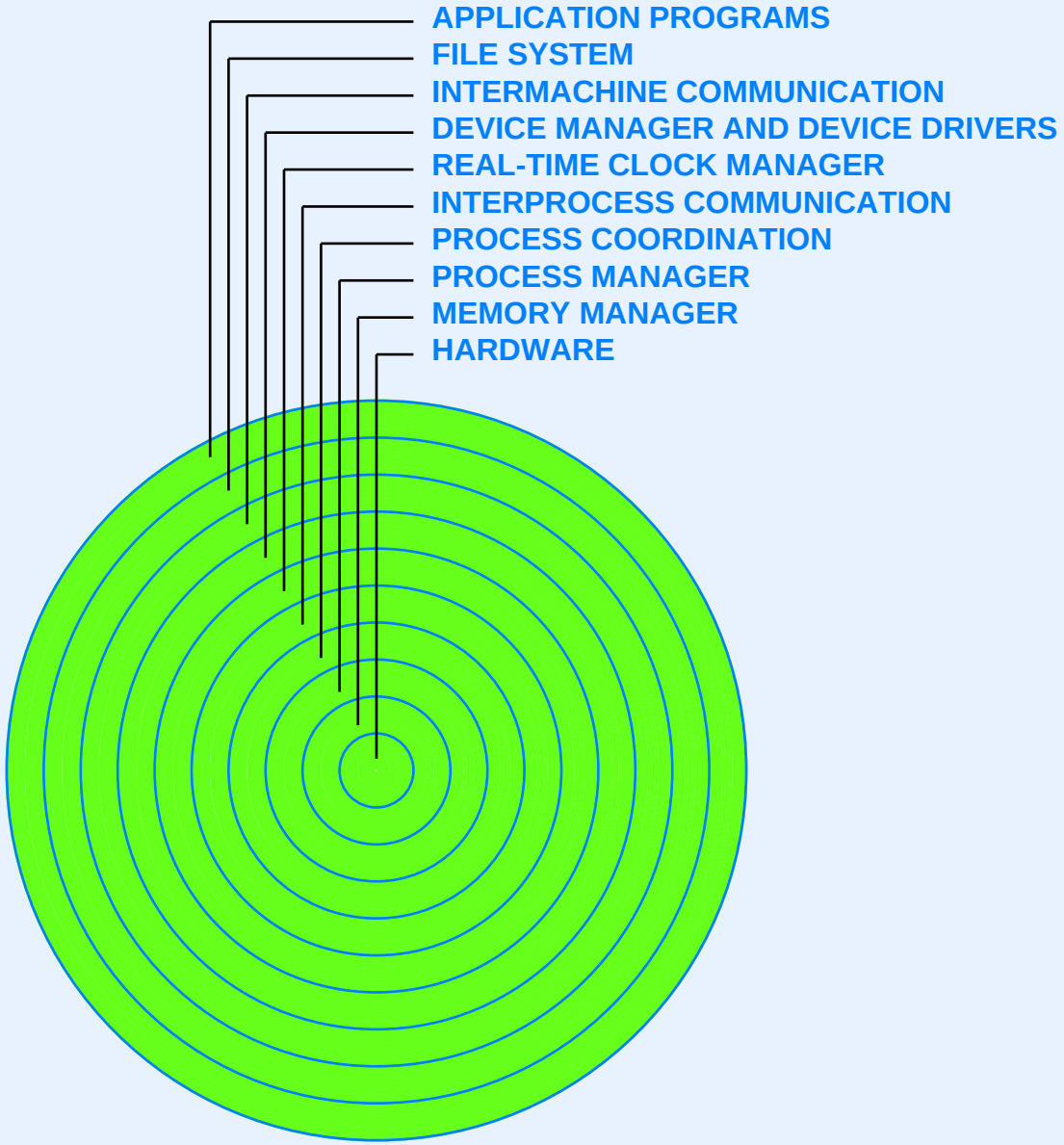
Multilevel Vs. Multilayered Organization

- Multilayer software
  - Visible to user as well as designer
  - Each layer uses layer directly beneath
  - Involves protection as well as data abstraction
  - Examples
    * OSI 7-layer model
    * MULTICS layered security structure
  - Can be inefficient

Multilevel Vs. Multilayered Organization (continued)

- Multilevel structure
  - Form of data abstraction
  - Used during system construction
  - Helps designer focus attention on one aspect at a time
  - Keeps policy decisions independent
  - Allows given level to use *all* lower levels

Multilevel Structure Of Xinu



How To Build An OS

- Begin with a *philosophy*
- Establish *policies* that follow the philosophy
- Design mechanisms that enforce the policies
- Implement mechanisms for specific hardware

Design Example

- Philosophy: “fairness”
- Policy: FCFS resource access
- Mechanism: queue of requests (FIFO)
- Implementation: program written in C

Queues And Lists

- Fundamental throughout an operating system
- Various forms
  - FIFOs
  - Priority lists
  - Ascending and descending order
  - Event lists ordered by time of occurrence
- Operations
  - Insert item
  - Extract “next” item
  - Delete arbitrary item

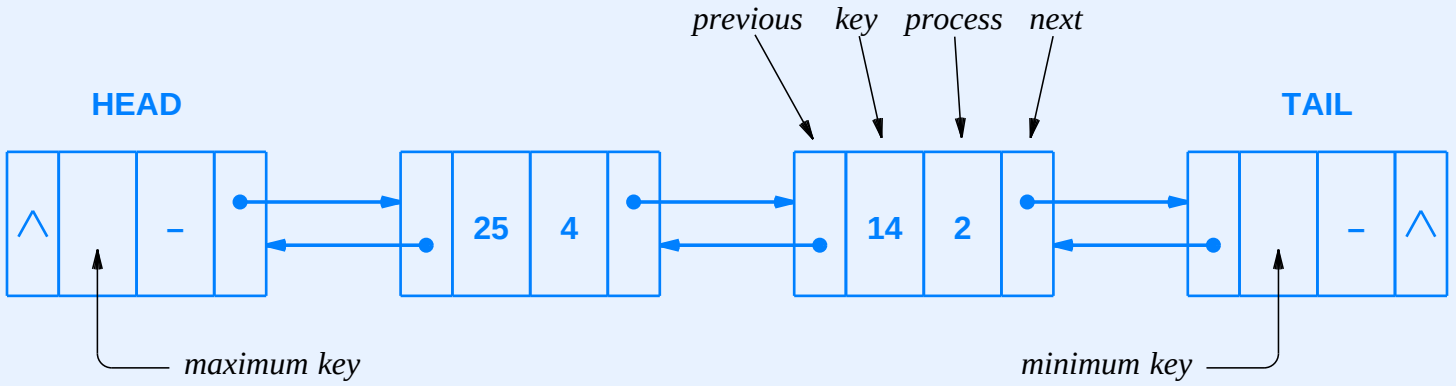
Lists And Queues In Xinu

- Fundamental ideas
  - Many lists store processes
  - A process is known by an integer *process ID*
  - Only need to store process's ID on list
- A single data structure can be used to store many types of lists

Unified List Storage

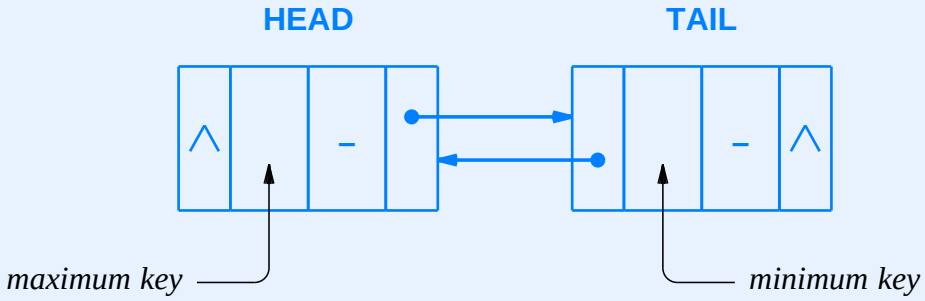
- All lists are doubly-linked, which means a node points to its predecessor and successor
- Each node stores a key as well as a process ID, even though a key is not used in a FIFO list
- Each list has a head and tail; the head and tail nodes have same shape as other nodes
- Non-FIFO lists are ordered in descending order
- The key value in a head node is the maximum integer used as a key, and the key value in the tail node is the minimum integer used as a key

Conceptual List Structure



- Example list contains two processes, 2 and 4
- Process 4 has key 25
- Process 2 has key 14

Pointers In An Empty List



- Head and tail linked
- Eliminates special cases for insertion or deletion

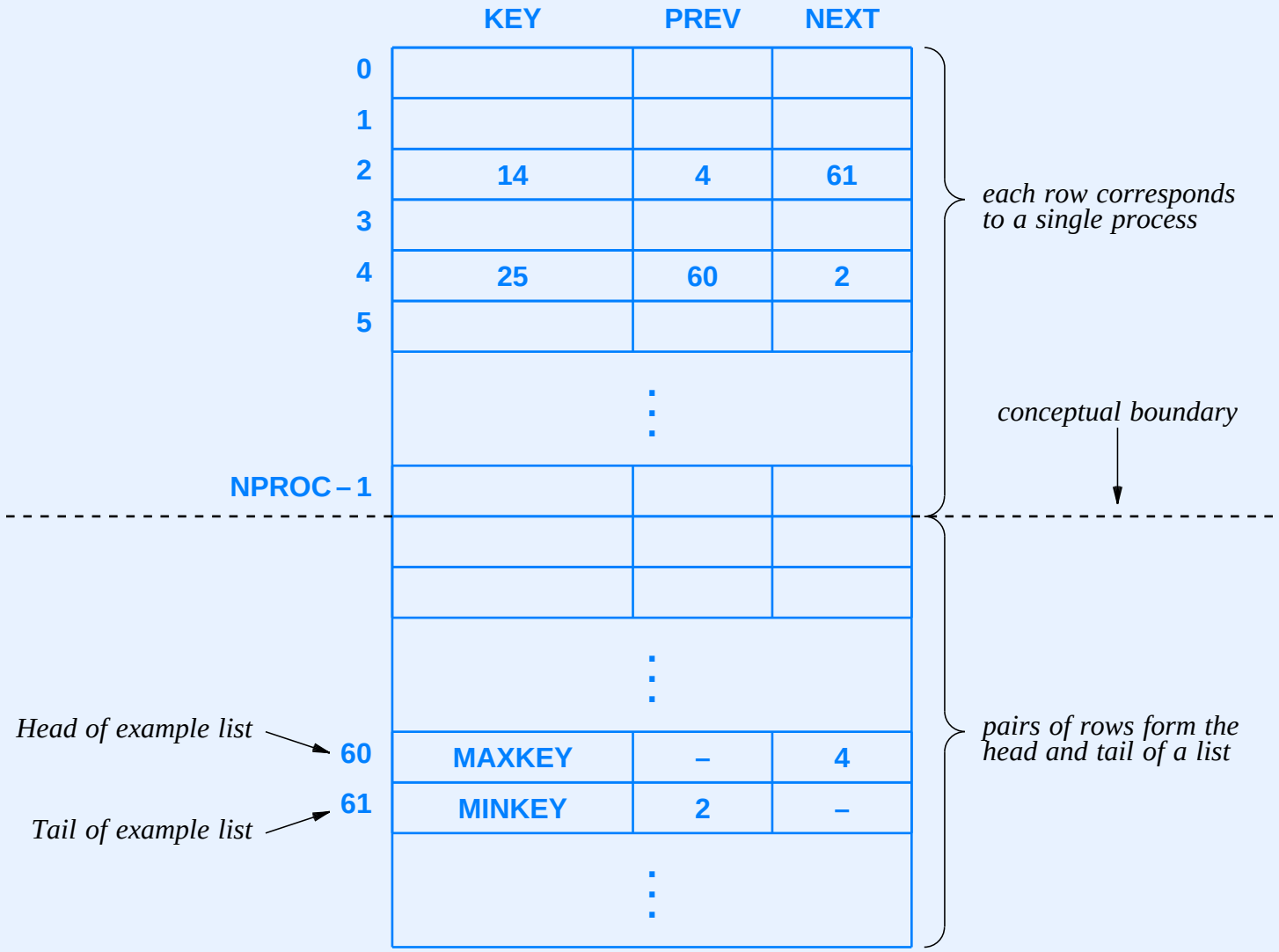
Reducing List Size

- Pointers can be expensive if a list is small
- Important concept: a process can appear on at most one list at any time
- Techniques used to reduce the size of Xinu lists
  - Relative pointers
  - Implicit data structure

Relative Pointers

- Store list elements in an array
  - Each item in array is one node
  - Use array index instead of address to identify a node
- Implicit data structure
  - Number processes 0 through *N - 1*
  - Let i^{th} element of array correspond to process *i*
  - Store heads and tails in same array at position *N* and higher

Illustration Of Xinu List Structure



Implementation

- Data structure
  - Consists of a single array named *queuetab*
  - Is global and available throughout entire OS
- Functions
  - Include tests, such as *isempty*, as well as insertion and deletion operations
  - Implemented with in-line functions when possible
- Example code shown after discussion of types

A Question About Types In C

- K&R C defines *short*, *int*, and *long* to be machine-dependent
- ANSI C leaves *int* as a machine-dependent type
- A programmer can define type names
- Question: should a type specify
  - The purpose of an item?
  - The size of an item?
- Example: should a process ID type be named
  - *processid_t* to indicate the purpose or
  - *int32* to indicate the size?

Type Names Used In Xinu

- Compromise to encompass both purpose and size
- Example: consider a variable that holds an index into queuetab
- The type name can specify
  - That the variable is a queue table index
  - That the variable is a 32-bit signed integer
- Xinu uses the type name *qid32* to specify both
- Example code follows

```
/* queue.h - firstid, firstkey, isempty, lastkey, nonempty                */

/* Queue structure declarations, constants, and inline functions          */

/* Default # of queue entries: 1 per process plus 2 for ready list plus */
/*                       2 for sleep list plus 2 per semaphore            */
#ifndef NQENT
#define NQENT   (NPROC + 4 + NSEM + NSEM)
#endif

#define EMPTY   (-1)            /* null value for qnext or qprev index  */
#define MAXKEY  0x7FFFFFFF      /* max key that can be stored in queue  */
#define MINKEY  0x80000000      /* min key that can be stored in queue  */

struct  qentry  {               /* one per process plus two per list    */
        int32   qkey;           /* key on which the queue is ordered    */
        qid16   qnext;          /* index of next process or tail        */
        qid16   qprev;          /* index of previous process or head    */
};

extern  struct qentry   queuetab[];

/* Inline queue manipulation functions */

#define queuehead(q)    (q)
#define queuetail(q)    ((q) + 1)
#define firstid(q)      (queuetab[queuehead(q)].qnext)
#define lastid(q)       (queuetab[queuetail(q)].qprev)
#define isempty(q)      (firstid(q) >= NPROC)
#define nonempty(q)     (firstid(q) <  NPROC)
#define firstkey(q)     (queuetab[firstid(q)].qkey)
#define lastkey(q)      (queuetab[ lastid(q)].qkey)
```

```
/* Inline to check queue id assumes interrupts are disabled */

#define isbadqid(x)     (((int32)(x) < 0) || (int32)(x) >= NQENT-1)

/* Queue function prototypes */

pid32   getfirst(qid16);
pid32   getlast(qid16);
pid32   getitem(pid32);
pid32   enqueue(pid32, qid16);
pid32   dequeue(qid16);
status  insert(pid32, qid16, int);
status  insertd(pid32, qid16, int);
qid16   newqueue(void);
```

Summary

- OS has well-understood components
- Complexity arises from interactions among components
- Multilevel approach helps organize system structure
- Design involves inventing policies and mechanisms that enforce overall goals
- Xinu includes a compact list structure that uses relative pointers and an implicit data structure to reduce size
- Xinu type names specify both purpose and size