論 文

OBJECT에 의한 운영체제의 구성에 대한 연구

正會員 李 昌 洙*

Object－Based Operating System

Chang Soo LEE*, *Regular Member*

要 約 본 논문은 신뢰성과 abstract data type을 효과적으로 뒷받침할 수 있도록 object에 기초를 둔 운영체제의 구성에 대하여 논하였다. 신뢰성을 위하여 모든 object는 capability에 의해서만 access가 가능하도록 access right을 규정하였으며 모든 program모듈은 protection domain을 갖도록 하여 이에 대한 효율적인 domain변경이 제시되었다. 또한 abstract data type을 위하여 type manager를 이용하였다.

ABSTRACT This paper describes object-based operating system to support reliability and abstract data type. For reliability, all objects should be accessed through access rights in capability, and the protection domain is provided for all program modules such that efficient domain switching can be achieved. For abstract data type, type manager is provided.

1. 서 론

크고 복잡한 소프트웨어 시스템을 만들고자할 때, 우리는 보통 이것을 몇 개의 비교적 간단한 subsystem으로 분할하여 구성하게 된다. 이 subsystem은 다시 하위의 subsystem으로 분할되며 이러한 분할은 더 이상의 분할 필요성이 없게 될 때까지 계속된다. 상위의 subsystem은 하위subsystem이 제공하는 abstraction을 이용하여 구성하게 된다. 이러한 방법의 장점은 상위 시스템은 하위의 implementation mechanism을 몰라도 되며, 따라서 모듈 단위의 incremental program development가 용이하게 된다[1].

여러 개의 모듈들이 하나의 시스템을 구성하게 될 때 모듈 상호간에는 어느 정도 interaction이 존재하게 된다. 이 경우 만약 어느 모듈에서 bug가 발생하였다면 이 bug는 다른 모듈로 파급되어 나갈 수도 있으며 이렇게 된다면 bug가 발생된 모듈이라든가 bug의 발생원인 등을 찾아낸다는 것이 매우 어렵게 될 것이다.

이러한 이유에서 error confinement를 위한 protection의 필요성이 절실해진다. 더우기 여러 사용자가 하나의 컴퓨터 시스템을 공유하여 사용하는 time-sharing system이라면 error propagation의 문제는 더욱 심각해진다.

또한 protection이 적절히 되지 않는다면 교묘한 사용자에 의한 중요한 정보의 유출을 억제하기 어렵다. 이와 같이 정보의 protection과 소프트웨어 신뢰성을 위하여 capability에 기초를 둔 memory addressing, 프로그램에 부여되는 최소한의 small protection domain, 그리고 data abstraction에 의한 extended-type object가 제안되었다[2].

한편, 운영체제는 종래의 process modal과 본 논문에서 살펴 보고자 하는 object model의 2종류로 나누어 생각할 수 있다[3].

object model의 O.S는 기본적으로 capability 를 관리하여 원하는 operation이 적절히 이루어질 수

* 東洋工業專門大學通信科
Dept. of Communication Engineering, Dongyang Junior Technical College, Seoul 150 Korea
論文番號 : 83- 04 (接受 1983. 1. 20)

있도록 하는 것이며 이것을 object-based operating system이라고 부르겠다. 본 논문에서 protection을 높이기 위하여 capability를 도입한 운영체제의 구성요소를 제시하였다. data abstraction에 의하여 발생되는 object들을 효율적으로 다루기 위하여 type manager를 이용하였다. 또한 간단한 프로그램에 적용하여 protection domain의 구성을 보이고 프로그램 수행에 따른 domain 변경방법을 제시하였다.

2. Objects

Object란 실제의 것이든 가상의 것이든 간에 어떤 resource의 abstraction에 의하여 발생하는 사물을 말한다.

Memory block, disk track, terminal controller, I/O port, tape등의 hardware resource는 물론, file, program, semaphore, directory등과 같은 software-created entity들이 모두 object라고 볼 수 있다[2].

하나의 object는 (unique-name, type, representation)의 집합으로 구성된다[4]

2·1 Unique-name

Processor #1, Processor #2, I/O Port #1, I/O Port #2등과 같이 모든 object는 각각을 구분할 수 있고 지칭할 수 있는 unique-name을 가지고 있다.

object의 지칭은 pointer approach와 unique identifier approach의 2종류로 나눌 수 있다[2].

Pointer approach는 object의 지칭이 pointer에 의하여 직접 이루어지므로 object에 도달하는 것은 쉬우나 object의 location이 dynamic하게 변경되는 경우 그 때마다 그 object에 대한 모든 capability를 변경하여야 한다는 단점이 있다.

Unique identifier approach는 모든 object에 unique name이 부여되어 있고 어떤 두 object도 같은 name을 가질 수 없다. Unique-name은 커다란 hash table에 저장되며 searching time을 줄이기 위하여 associative memory를 두는 것이 보통이다. Unique identifier approach의 장점은 object의 location에 따른 capability의 변경이 불필요하다는 것이며 단점으로는 hash table을 위한 space와 time을 들 수 있다.

여기서 Unique-name은 unique identifier approach를 채택한 것이다.

2·2 Type

Type은 object가 속하고 있는 class의 형태를 나타내어 준다. Data가 정수형, 실수형 등의 type을 가지고 있듯이 각object는 File, Device, Procedure, Port, Manager등의 type을 갖게 된다. 이러한 object의 type에 따라 object에 적용할 수 있는 operation이 결정된다. 보통 여러 object 들이 동일한 type을 갖게 된다.

System-defined type외에 user가 새로운 type을 정의하여 사용할 수도 있으며 이것을 extended type 혹은 abstract data type라 부른다.

Type의 내용은 다른 object의 unique name이며, 이 object가 type manager이다. abstract data에 대한 operation은 type manager에 의해 수행되며 type manager는 type checking을 행한 후 원하는 operation에 대한 access right가 있는 경우에만 수행한다.

2·3 Representation

Representation은 data part와 capability part로 구성된다. Data part에는 instruction 혹은 data가 저장된다.

Capability part에는 access할 수 있는 object의 unique name과 그 object에 허용되는 access right로 구성되는 capability들이 저장된다[4].

Access right는 system right와 auxiliary right로 구분된다. System right에는 Read, Write, Execute, Enter, Delete, Copy, Share, Pass등의 것을 들 수 있다. Execute right은 capability의 unique-name이 지칭하는 object에 대하여 instruction fetch를 허용하는데 사용된다.

Enter right는 다른 domain에 속하는 procedure를 reference한다는 의미이며 domain switching을 필요로 한다. Delete right는 capability의 삭제를 허용하기 위하여 필요하다. Copy right은 object의 내용을 다른 object에 복사하는 경우에 필요하며, Share right는 그 object에 대한 capa를 다른 object도 가질 수 있도록 하는 경우에 사용된다. Pass right는 다른 domain에 capability를 넘겨주는 것을 허용하는데 사용된다.

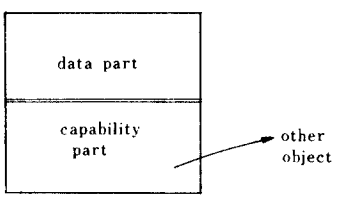

그림 1 Representation구조
Representation structure.

Copy right의 checking방법은 object의 data를 읽을 때 copy right가 없는 것이 발견되었으면 object의 unique-name과 copy-not-allowed flag을 set시켜 data를 저장하려 할 때 대상object의 unique name과 비교하여 두 object의 name이 일치하는 경우에만 허용한다.

다른 object에 capability를 넘겨 줄 때에는 access right를 현재의 것 이상을 허용해서는 안된다. Child capability는 parent capability의 access right보다 더 클 수 없다.

Auxiliary right는 abstract data에 대한 type manager를 수행할 경우에 적용된다. Type manager가 제공하는 operation 각각에 대하여 right를 설정하여 두고 어떤 operation을 요구하였을 때 이 auxiliary right의 해당부분을 검사하게 된다.

2·4 Short-term State (일시적 상태)

object의 data part가 instruction segment이고 active상태에 있는 경우에는 short-term state가 필요하다. 이에는 instruction pointer, stack pointer, status information, domain index, calling procedure에 대한 capability, local data segment에 대한 capability, parameter에 대한 capability등이 속한다.

해당object가 실제 processor에서 running 중인 경우라면 이것은 실제 processor의 내용이 될 것이며 calling procedure의 short-term state는 stack에 저장된다.

3. Memory Addressing

Object를 저장하고 있는 primary memory의 addressing방법으로 capability-based addressing을 이용한다[2],[5].

Capability-based addressing은 object의 proctection과 sharing을 용이하게 하며 reliable software와 system security를 위한 효과적인 방법으로 주장되어 왔다. 이것의 단점으로는 mapping을 위한 space와 time이 소요된다는 것을 들 수 있다.

Object는 segment단위로 저장되며 segment type은 저장되는 object type에 의해 결정된다.

Segment내의 한 word는 segment에 대한 identifier (capability의 unique name)와 segment내의 offset에 의해 번지지정이 된다. Logical address는 이와 같이 (unique name, displacement)의 쌍으로 이루어지는데 physical addresss로의 변환은 capability list와 CMT (Central Mapping Table)의 두 mapping table에 의하여 그림 3과 같이 이루어진다[6]. Capability list는 그 domain에서 access할 수 있는 모든 object에 대한 capability가 들어 있다. CMT는 특정object가 primary memory에 있는가의 여부, physical address 혹은 disk address, object type, memory management를 위한 정보 등을 가지고 있는 hash table이다.

그림 3과 같이 two-level mapping을 행하므로 object location에 무관하게 object sharing이 용이하며 (그림 4), access시 capability list의 capability에 따라 object reference가 허용되므로 sharing object에 수행 가능한 operation을 달리하기가 용이하다. 반면 memory reference시 mapping 및 searching overhead를 줄이기 위하여 소규모의 associative memory를 두는 것이 보통이다.

메모리에 저장되어 있는 내용이 data이냐 capability이냐를 구별하는 방법으로는 tagged approach와 partition approach의 두 가지가 있다[5]. Tagged approach는 segment내의 각 word에 대해 여분의 tag bit를 두어 구별하는 방법이고 partition approach는 capability를 저장하는 segment와 data

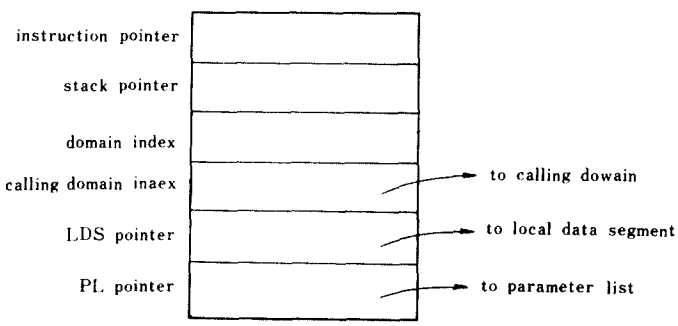

그림 2 일시적 상태
Short-term state.

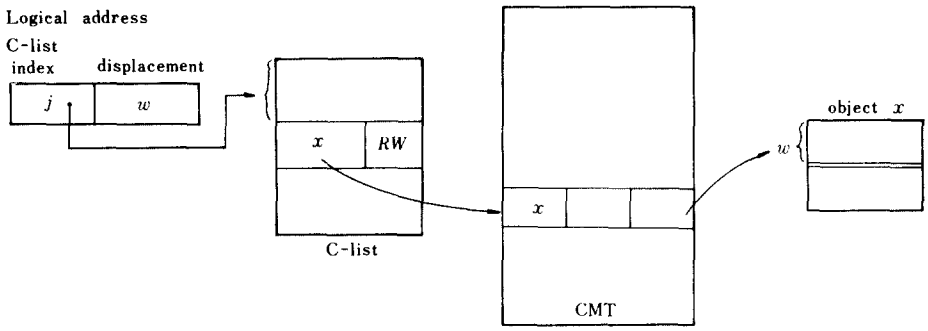

그림 3 Mapping 과정
Mapping mechanism.

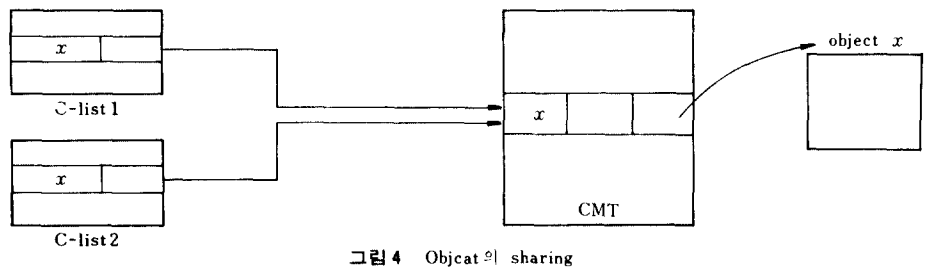

그림 4 Objcat의 sharing
Sharing of object.

를 저장하는 segment를 구분하여 사용하는 방법이다.

한 segment내에 data와 capability를 병용할 수 있는 tagged approach가 program structure에 더 가깝다.

4. Protection Domain

Software reliability와 system security를 위하여 hardware/software의 fault나 malicious user의 그릇된 접근으로부터 보호를 할 수 있는 어떤 protection mechanism이 있어야 한다.

종래의 제 3 세대 computer는 user software error가 다른 user나 system에 영향을 주는것을 막기 위하여 privileged supervisor state와 nonprivileged problem state를 두어 protection을 이루려 했다. 그러나 privileged state에 속하는 O. S. 는 모든 권한을 부여받았기 때문에 만약 privileged state에 속하는 operating system에서 error가 발하였을 경우 이것을 protection할 방법이 없었다. 또한 system을 잘 알고 있는 악의의 사용자가 privileged state를 교묘하게 이용할 수 있는 여지를 남게 놓았다.

이것의 해결책으로 한 program이 access할 수 있는 부분을 제한하여 허용된 범위를 벗어나는 행동은 일어날 수 없도록 제한하는 방법이 주장되어 왔다. 모든 program은 그것이 access할 수 있는 object들이 정해져 있으며 그 object들에 허용된 access right도 제한되어 있다. 어떤 program이 접근할 수 있는 object들과 이들에 대한 access right의 집합을 protection domain이라 부른다. 이러한 protection domain에 대하여 최소의 access right를 부여했을 때 small protection domain 이라 부른다[2].

하나의 program은 여러 개의 subunit로 구성되는 것이 일반적이며 subunit들은 그들마다의 protection domain이 주어져 있어 전체 program 수행을 위하여는 이들 protection domain을 오가야 할 필요가 있다. 이와 같이 protection domain의 변경은 object-based system의 특징으로 볼 수 있으며 이것을 domain switching이라 부른다.

5. Domain Switching

다른 domain에 속하는 procedure의 호출이나 abstract data에의 operation수행을 위한 type manager의 호출이 있게 되면 domain switching을 하게 된다. Domain switching시 문제가 되는 것은 short-term state의 보존과 parameter의 전달방법이다. Domain변경시에는 현domain의 short-term

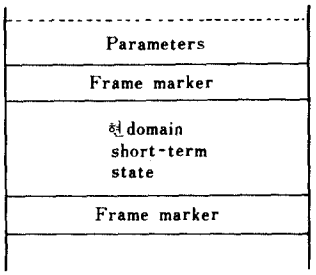

그림 5 Procedure 호출시의 stack
Stack before procedure calling.

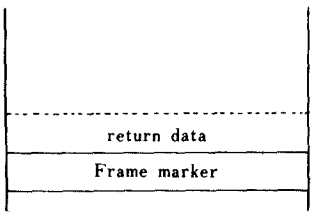

그림 6 Procedure로 부터 복귀시의 stack
Stack after procedure return.

state를 stack에 저장하고 그 위에 frame marker를 붙인 다음 parameter를 저장한다 (그림 5).

Called procedure에서의 복귀는 저장된 short term state를 원 상태로 하고 return data 혹은 capability를 stack에 저장하는 것에 의하여 이루어진다 (그림 6).

Type manager domain으로의 변경시에는 parameter외에 수행할 operation name과 access right를 stack을 통하여 전달한다.

Type manager에서는 right checking을 행한다.

Type manager의 호출은

Type call (type name, operation select, parameters)와 같은 명령에 의해 이루어진다.

보통 domain 변경은 빈번히 발생하므로 효율면에서 볼 때 hardware level에서 domain switching이 이루어질 수 있어야 한다.

6. Interprocess Communication

Asynchronous process간의 communication은 PORT를 통하여 이루어진다. Communication되는 information은 message로 구성되며 message object라 부르며 어떤 object도 message로 될 수 있다.

PORT object는 정해진 수의 output channel, input channel, message queue, blocked process queue로 구성된다. message 는 output channel로 나가고 input channel로 들어온다.

빈 message queue에 대하여 SEND를 하고자 하였을 때 명령을 발한 process는 blocked process queue로 들어간다. Message queue에는 message object에 대한 capability들이 저장된다[4].

7. 적 용 예

본 절에서는 object와 domain의 구성, 그리고 program이 수행될 때 domain switching에 의한 domain의 변화상태를, 참고문헌[1]에 있는 count-words program에 적용하여 본다. 프로그램은 부록에 첨부하여 놓았다.

Count-words procedure의 하는 일은 instream type의 input text로부터 단어를 찾아내어 어떤 단어가 몇 번 나왔는가를 세어서 outstream type의 object에 print하는 것이다.

이 프로그램이 수행될 때의 CMT내용은 그림7과 같이 될 것이다.

Count-words procedure가 수행되기 위한 domain의 구성은 그림 8과 같다.

D register의 내용은 object의 unique name과 C-list가 시작되는 offset가 되겠다. LDS와 PL 내 특정date 혹은 capability는 LDP 혹은 PLP의 내용과 displacement의 합이 유효번지로 되는 index addressing에 의하여 지정된다.

Count-words내의 wordbag$create( )라는 명령에 의해 wordbag의 type manager로 domain switching이 일어나게 된다. 우선 type checking을 행한 후 현재의 processor register내용이 stack 어

Unique-name	Type	Descriptor
count-words	Proc	
next-word	〃	
wordbag	Manager	
wordtree	〃	
instream	〃	
outstream	〃	
wb	Wordbag	
w	string	
contents	wordtree	
total	integer	
j	instream	
o	outstream	

그림 7 Central mapping table의 내용
Contents of central mapping table.

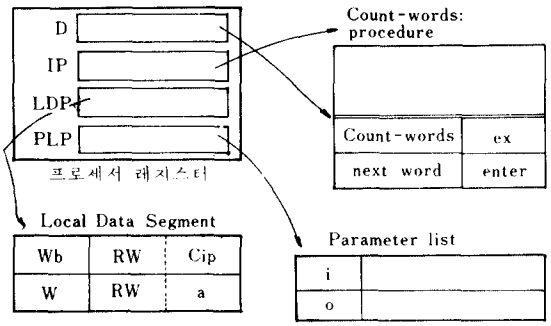

그림 8 Count-words 수행시의 domain
Domain of count-words procedure.

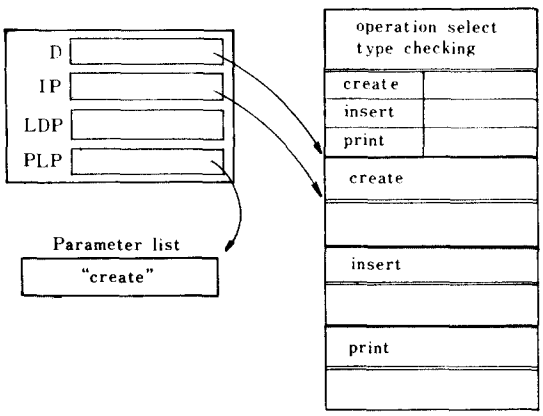

그림 9 Wordbag 수행시의 domain
Domain of wordbag type manager.

저장되며 frame marking을 한 후 parameter가 그 위에 저장된다. Wordbag$create의 경우는 parameter가 없다.

Called domain의 parameter list로 옮겨지고 pointer에 값이 지정된다. Type manager에서는 원하는 operation을 선택하고 auxiliary right을 check하게 된다. System right도 type manager에서 필요한 만큼 abstract data의 capability에 표시되어야 한다. 그림 9에 wordbag type manager를 수행할 때의 domain을 표시하였다.

한편 wordbag에서는 wordbag type의 abstract data를 cvt라는 명령에 의해 representation type으로 변환하여 다루게 되는데 이것은 그림 10에서 보는 바와 같이 wordbag type의 wb에 대한 capability대신 wordtree type의 contents와 integer type의 total로 분할하여 사용하게 되는 것을 의미한다. rep type으로부터 역cvt는 wb에 대한 capability를 갖도록 하는 것을 의미한다.

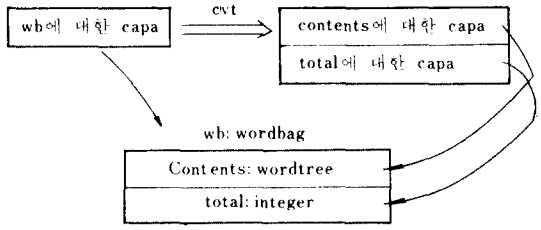

그림 10 Cvt에 의한 representation 형식으로의 변화
Transformation of abstract data by "cvt"

8. 결 론

운영체제의 구성에 capability를 도입하여 신뢰성을 높일 수 있으며 data abstraction의 처리는 type manager를 이용함으로써 일관성을 이룰 수 있다. domain의 변경이 빈번히 발생하므로 효율적인 처리를 위하여 하아드웨어에 의하여 이루어져야 하겠다. 또한 object에 기초에 두어 설계를 한다면 high-level machine을 이룰 수 있는 가능성이 있다.

9. 부록 (문장내의 단어를 세는 프로그램)

```
count words = proc (i: instream,  o: outstream);
  % create an empty wordbag
  wb: wordbag: = wordbag$create ( );
  %scan document adding each word found to wb
  w: string: = next word (i);
  while w = " " do
    wordbag$insert (wb, w);
    w: next word (i);
    end;
% print the wordbag
wordbag$print (wb, o);
end count words;

next word = proc (i: instream) returns (string);
  c: char: = " ";
  % scan for first alphabetic character
  while alpha (c) do
    if instream$empty (i)
      if inst        " ";    then return
      end;
    c: = instreamSnext (i);
    end;
  % accumulate characters in word
  w: string: = " ";
  while alpha (c) do
    w: = string$append (w, c);
    if instream$empty (i)
```

```
      then return (w);
      end;
    c: =instream$next (i);
    end;
  return (w); % the nonalphabetic character c is lost
  end next word;

wordbag=cluster is
  create,   % create an empty bag
  insert,   % insert an element
  print,    % print contents of bag
  rep=record [contents: wordtree,  total: int] : ;
create=proc (  ) return (cvt);
        return (rep${ contents: wordtree$create (  ),
        total: 0} ;
        end create;
insert=proc (x; cvt, v: string);
        x. contents: =wordtree$insert (x. contents, v);
        x. total: =x. total+1;
        end insert;
print=proc (x: cvt, o: outstream);
       wordtree$print (x. contents,  x. total, o);
       end print;
end wordbag;

wordtree=cluster is
  create,   % create empty contents
  insert,   % add item to contents
  print;  % print contents
  node=record [value: string,  count: int,  lesser: word-
       tree,  greater: wordtree];
  rep=oneof [empty: null, non empty: node];
create=proc (  ) returns (cvt);
  return (rep$make empty (nil));
  end create;
insert=proc (x: cvt, v: string)  returns (cvt);
  tagcase x
    tag empty:
      n: node: =node${ value: v, count: 1, lesser: word-
                tree$create (  ), greater: wordtree$
                create (  ) } ;
      return (rep$make non empty (n));
    tag non empty (n: node) :
      if v=n. value
            then n. count: =n. count+1;
        elseif v<n. value
          then n. lesser: =wordtree$insert (n. lesser,
                          v);
        else n. greater: =wordtree$insert (n. greates,
                         v);
        end;
      end;
    return (x);
  end insert;
print=proc (x: cvt, total: int, o: outstream);
  tagcase x
    tag empty: ;
    tag non empty (n: node);
      wordtree$print (n. lesser, total, o);
        print word (n. value,  n. count, total, o);
      end;
    end print;
  end wordtree;
```

參 考 文 獻

(1) B. Liskov, et al., "Abstraction mechanisms in CLU," C. A. C. M., vol. 20, no. 8, pp. 564－576, Aug. 1977.

(2) T. A. Linden, "Operating system structures to support security and reliable software," Computing Surveys, vol. 8, no. 4, pp. 409－445, Dec. 1976.

(3) A. S. Tanenbaum, "Computer networks," Prentice-Hall, Inc., 1981.

(4) W. Wulf, et al., "HYDRA/C. mmp: An experimental computer system," McGraw-Hill, Inc., 1981.

(5) R. S. Fabry, "Capability-based addressing," C. A. C. M., vol. 17, no. 7, pp. 403－412, July 1974.

(6) P. J. Denning, "Fault tolerant operating systems," Computing Surveys, vol. 8, no. 4, pp. 359－389, Dec. 1976.

李 昌 洙 (Chang Soo LEE)　正會員

1953年 11月 30日生

1976年 2月：서울大學校工科大學電氣科 卒業

1983年 8月：서울大學校工科大學大學院 電算科碩士過程卒業豫定

1976年～1981年：東洋精密中央研究所 先任研究員

1981年 9月～現在：東洋工業專門大學通信科專任講師