บทที่ 6

การประยุกต์ใช้งานและข้อเสนอแนะ

การประยุกต์ใช้งาน

เพื่อเป็นการทดสอบการทำงานของไลบรารีฟังก์ชันที่พัฒนาขึ้น จึงนำตัวอย่างโปรแกรมดังแสดงในบทที่ 2 มาดัดแปลงเป็นภาษาซีตามหลักการประยุกต์ในบทที่ 3 อย่างไรก็ตาม เนื่องจากเครื่องที่ทำการทดสอบมีข้อจำกัดเรื่องจำนวนโปรเซสต่อผู้ใช้แต่ละคน จำนวนโปรเซสที่ใช้ในโปรแกรมทดสอบนี้จึงใช้เพียง 12 โปรเซสเท่านั้น คือ sieve(0) ถึง sieve(10) และ print() และ โปรแกรมนี้จะหาค่า prime number ที่อยู่ในช่วง 1 ถึง 60 เท่านั้น

โปรแกรมที่ทำการทดสอบมีเนื้อหาดังนี้

```
1   #include "csp.h"
2   int sieve();
3   int print();
4   struct funcMap funcMap[] = {
5       {"sieve", sieve},
6       {"print", print},
7       {NULL, 0}
8   };
9
10  /* ------------------------------------------------------------ */
11  /* main program                                                 */
12  /*    - initialize pointer to array of pointers to function     */
13  /*      names to be executed                                    */
14  /*    - call function cspPar to fork and execute the functions*/
15  /* ------------------------------------------------------------ */
16  main()
17  {
```

```
18        static char *param[] = {"sieve(0)","print()","sieve(i:1..5)",
19                                "sieve(6)", "sieve(7)", "sieve(8)",
20                                "sieve(9)", "sieve(10)", NULL};
21        if (cspPar(param) == ERROR)
22            printf("Parallel failed\n");
23    }
24    /* ------------------------------------------------------------ */
25    /* This function is separated into 3 parts                      */
26    /*      - sieve(0)                                              */
27    /*            first, output number 2 to print                   */
28    /*            then,  begin with number 3 and loop to send       */
29    /*                   odd number to sieve(1) until the number    */
30    /*                   is greater than MAX_NUM                    */
31    /*      - sieve(10)                                             */
32    /*            loop to receive data from sieve(9) until          */
33    /*            sieve(9) is terminated                            */
34    /*      - sieve(1) thru sieve(9)                                */
35    /*            first, input one number from the lower sieve      */
36    /*                   process and send that number to print      */
37    /*            then,  loop to input data from the lower sieve    */
38    /*                   and check to see if that number can be     */
39    /*                   divided by the first one with no remainder*/
40    /*                         if yes, by pass it                   */
41    /*                         if no, send it to the next sieve     */
42    /*                   this process will continue to do this      */
43    /*                   until the lower sieve is terminated        */
44    /* ------------------------------------------------------------ */
45    int sieve(i)
46    int i;
47    {
```

```
48      int j;
49      switch(i) {
50          case 0:
51          {
52              int n;
53              n = 2;
54              cspOut("print()", &n);
55              n = 3;
56              while (TRUE)
57              {
58                  if (n < MAX_NUM)
59                  {
60                      cspOut("sieve(1)", &n);
61                      n += 2;
62                  }
63                  else
64                      break;
65              }
66              break;
67          }
68          case 10:
69              while (TRUE)
70              {
71                  int n;
72                  int ret_code;
73                  ret_code = cspIn("sieve(9)", &n, IPC_NOWAIT);
74                  if ((ret_code==COMM_ERR)||(ret_code==COMM_DIE))
75                      break;
76                  else
77                      if (ret_code == COMM_OK)
78                          cspOut("print()", &n);
```

```
79              }
80              break;
81          default:
82              int p, mp;
83              if (cspIn("sieve(i-1)", &p, 0) == COMM_OK)
84              {
85                  cspOut("print()", &p);
86                  mp = p;
87                  while (TRUE)
88                  {
89                      int m;
90                      int ret_code;
91                      ret_code = cspIn("sieve(i-1)", &m, 0);
92                      if ((ret_code==COMM_NOK)||(ret_code==COMM_DIE))
93                          break;
94                      else
95                      {
96                          while (TRUE)
97                          {
98                              if (m > mp)
99                                  mp += p;
100                             else
101                                 break;
102                         }
103                         if (m = mp)
104                             ;
105                         else
106                             if (m < mp)
107                                 cspOut("sieve(i+1)", &m);
108                             else
109                                 ;
```

```
110                         }
111                     }
112                 }
113         }
114 }
115 /* ------------------------------------------------------------ */
116 /* print function loop to receive data from sieve(0) to        */
117 /* sieve(10) until all of the sieve processes are terminated   */
118 /* (flag p_alive is FALSE)                                     */
119 /* ------------------------------------------------------------ */
120 int print()
121 {
122     while (TRUE)
123     {
124         int i, n;
125         int p_alive;
126         char p_name[10];
127         for (i=0; i<=9; p_name[i++]=0);
128         strcpy(p_name, "sieve(0)");
129         p_alive = FALSE;
130         for (i=0; i<=10; i++)
131         {
132             if (i==10)
133                 strcpy(p_name,"sieve(10)");
134             else
135                 p_name[6] = i + '0';
136             switch(cspIn(p_name, &n, IPC_NOWAIT)
137             {
138                 case COMM_OK:
139                     printf("prime number = %d\n", n);
140                     if (!(p_alive))
```

```
141                             p_alive = TRUE;
142                         break;
143                     case COMM_NOK:
144                         if (!(p_alive))
145                             p_alive = TRUE;
146                         break;
147                     case COMM_DIE:
148                         break;
149             }
150         }
151         if (!(p_alive))
152             break;
153     }
154     printf("job completed\n");
155 }
```

<u>คำอธิบายโปรแกรม</u>

บรรทัดที่ 1 โปรแกรมประยุกต์ต้องกำหนดบรรทัดนี้เพื่อเรียกใช้แฟ้มข้อมูล (file) (วิศวกรรมสถานแห่งประเทศไทย, 2527-2530) ที่กำหนดตัวแปรร่วม แฟ้มข้อมูลนี้ชื่อ csp.h (ดูภาคผนวก)

บรรทัดที่ 2 ประกาศว่าฟังก์ชัน sieve เป็นฟังก์ชันที่ส่งค่ากลับเป็นเลขจำนวนเต็ม

บรรทัดที่ 3 ประกาศว่าฟังก์ชัน print เป็นฟังก์ชันที่ส่งค่ากลับเป็นเลขจำนวนเต็ม

บรรทัดที่ 4-8 โปรแกรมประยุกต์ต้องกำหนดตัวแปรให้มีโครงสร้างเหมือนโครงสร้าง funcMap ซึ่งกำหนดไว้ในแฟ้มข้อมูล csp.h เพื่อบอกว่าถ้าต้องการประมวลผลโปรเซสชื่อใดให้เรียกประมวลผลฟังก์ชันใด

ในโปรแกรมทดสอบนี้ใช้ตัวแปรชื่อ funcMap และ กำหนดให้ว่าถ้ามีการเรียกประมวลผลโปรเซสชื่อ sieve จะต้องประมวลผลฟังก์ชัน sieve ซึ่งส่งค่ากลับเป็นเลขจำนวนเต็ม และ ถ้ามีการเรียกประมวลผลโปรเซสชื่อ print จะต้องประมวลผลฟังก์ชัน print ซึ่งส่งค่ากลับเป็นเลขจำนวนเต็มเช่นกัน ส่วนค่า NULL เป็นการบอกการสิ้นสุดการกำหนดค่าในตัวแปรนี้

บรรทัดที่ 10-15 เป็นหมายเหตุ (comment) ของโปรแกรม

บรรทัดที่ 16-17 ส่วนเริ่มต้นของโปรแกรม

บรรทัดที่ 18-20 กำหนดให้ตัวแปรชื่อ param เป็นตัวแปร 2 มิติ ที่เก็บชื่อโปรเซสต่าง ๆ ที่ต้องการประมวลผล โดยโปรเซสแรกคือ sieve(0) โปรเซสที่ 2 คือ print() โปรเซสที่ 3 ถึง 7 คือ sieve(1), sieve(2), sieve(3), sieve(4) และ sieve(5) โปรเซสที่ 8 คือ sieve(6) โปรเซสที่ 9 คือ sieve(7) โปรเซสที่ 10 คือ sieve(8) โปรเซสที่ 10 คือ sieve(9) และ โปรเซสสุดท้าย คือ sieve(10) สำหรับค่า NULL เป็นการบอกจุดสิ้นสุดการกำหนดค่าให้ตัวแปรนี้

บรรทัดที่ 21-22 เรียกฟังก์ชันซีเอสพีพาร์ โดยส่งค่าของตัวแปรชื่อ param เป็นค่าพารามิเตอร์ และ ตรวจสอบว่าค่าที่ส่งกลับมาเท่ากับค่าคงที่ ERROR (ซึ่งกำหนดในแฟ้มข้อมูล csp.h) หรือไม่ ถ้าเท่า จะพิมพ์คำว่า Parallel failed ที่หน่วยแสดงผล (ปกติจะเป็นจอภาพ)

บรรทัดที่ 23 จุดสิ้นสุดของโปรแกรม

บรรทัดที่ 24-44 เป็นหมายเหตุของโปรแกรม

บรรทัดที่ 45-47 จุดเริ่มต้นของฟังก์ชัน sieve ที่จะถูกประมวลผล ซึ่งสามารถรับค่าพารามิเตอร์ที่เป็นเลขจำนวนเต็มได้ 1 ค่า

บรรทัดที่ 48 กำหนดตัวแปร j เป็นชนิดเลขจำนวนเต็ม

บรรทัดที่ 49 ทดสอบค่าพารามิเตอร์ที่รับมา ถ้าเป็น 0 จะประมวลผลบรรทัดที่ 50 ถึง 67 ถ้าเป็น 10 จะประมวลผลบรรทัดที่ 68 ถึง 80 มิฉะนั้น จะประมวลผลบรรทัดที่ 81 ถึง 113

บรรทัดที่ 52-53 กำหนดตัวแปร n เป็นชนิดเลขจำนวนเต็ม มีค่าเท่ากับ 2 ตัวแปรนี้จะใช้ได้ภายในบรรทัดที่ 51 ถึง 67 เท่านั้น

บรรทัดที่ 54 เรียกฟังก์ชันซีเอสพีเอาท์ เพื่อส่งค่าในตัวแปร n ให้โปรเซสชื่อ print

บรรทัดที่ 55 กำหนดให้ตัวแปร n มีค่าเท่ากับ 3

บรรทัดที่ 56-65 เป็นการประมวลผลการทำซ้ำ โดยทำการตรวจสอบว่าค่าในตัวแปร n น้อยกว่าค่าคงที่ MAX_NUM (ซึ่งกำหนดไว้ในแฟ้มข้อมูล csp.h ให้มีค่าเท่ากับ 60) หรือไม่ ถ้าน้อยกว่า จะเรียกฟังก์ชันซีเอสพีเอาท์เพื่อส่งค่าในตัวแปร n ให้โปรเซส sieve โดยมีค่าพารามิเตอร์เป็น 1 และ เพิ่มค่าในตัวแปร n ขึ้นอีก 2 แล้วกลับไปทดสอบค่าในตัวแปร n ใหม่ แต่ถ้าค่าในตัวแปร n มากกว่าหรือเท่ากับค่าคงที่ MAX_NUM จะหลุดจากการประมวลผลการทำซ้ำนี้

บรรทัดที่ 69-79 เป็นการประมวลผลการทำซ้ำ โดยกำหนดตัวแปร n และ ret_code เป็นเลขจำนวนเต็มซึ่งเป็นที่รู้จักในการประมวลผลการทำซ้ำช่วงนี้เท่านั้น จากนั้น

จะเรียกฟังก์ชันซีเอสพีอินเพื่อรับค่าจากโปรเซส sieve ที่มีพารามิเตอร์เป็น 9 มาไว้ในตัวแปร n ทั้งนี้ การรับค่าจะเป็นแบบไม่รอ ค่าที่ส่งกลับมาจากฟังก์ชันซีเอสพีอินจะถูกเก็บในตัวแปร ret_code ซึ่งถ้ามีค่าเท่ากับค่าคงที่ COMM_ERR หรือ COMM_DIE (คือไม่สามารถรับค่าข้อมูลได้) จะหลุดจากการประมวลผลการทำซ้ำนี้ แต่ถ้ามีค่าเท่ากับค่าคงที่ COMM_OK (คือรับค่าข้อมูลได้) จะเรียกฟังก์ชันซีเอสพีเอาท์เพื่อส่งค่าในตัวแปร n ให้โปรเซส print แล้วประมวลผลซ้ำรอบถัดไป

บรรทัดที่ 82 กำหนดตัวแปรชื่อ p และ mp เป็นชนิดเลขจำนวนเต็ม

บรรทัดที่ 83-86 เรียกฟังก์ชันซีเอสพีอินเพื่อรอรับค่าจากโปรเซส sieve ที่มีค่าพารามิเตอร์น้อยกว่าค่าพารามิเตอร์ของโปรเซสที่ประมวลผลนี้อยู่ 1 ค่าที่ได้จะเก็บในตัวแปร p ถ้าสามารถรับค่าได้จะเรียกฟังก์ชันซีเอสพีเอาท์เพื่อส่งค่าในตัวแปร p ให้โปรเซส print จากนั้น กำหนดให้ตัวแปร mp มีค่าเท่ากับตัวแปร p

บรรทัดที่ 87-111 เป็นการประมวลผลการทำซ้ำ โดยกำหนดตัวแปร m และ ret_code เป็นชนิดเลขจำนวนเต็ม (ซึ่งรู้จักภายในการประมวลผลการทำซ้ำนี้เท่านั้น) จากนั้นเรียกฟังก์ชันซีเอสพีอินเพื่อรอรับค่าจากโปรเซส sieve ที่มีค่าพารามิเตอร์น้อยกว่าค่าพารามิเตอร์ของโปรเซสนี้อยู่ 1 ค่าที่ได้จะเก็บในตัวแปร m ถ้าไม่สามารถรับค่าได้ (ค่าที่ส่งกลับมามีค่าเท่ากับค่าคงที่ COMM_NOK หรือ COMM_DIE) จะหลุดจากการประมวลผลการทำซ้ำนี้ แต่ถ้ารับค่าได้ จะเพิ่มค่าตัวแปร mp ด้วยค่าในตัวแปร p จนกว่าค่าในตัวแปร m น้อยกว่าหรือเท่ากับค่าในตัวแปร mp จากนั้น ถ้าค่าในตัวแปร m น้อยกว่า mp จะเรียกฟังก์ชันซีเอสพีเอาท์เพื่อส่งค่าในตัวแปร p ให้โปรเซส sieve ถัดไป และ กลับไปเริ่มการประมวลผลซ้ำต่อไป

บรรทัดที่ 114 บอกจุดสิ้นสุดของฟังก์ชัน sieve

บรรทัดที่ 115-119 เป็นหมายเหตุของโปรแกรม

บรรทัดที่ 120-121 บอกจุดเริ่มต้นฟังก์ชัน print

บรรทัดที่ 122-153 เป็นการประมวลผลการทำซ้ำ โดยมีการกำหนดตัวแปร i ตัวแปร n และ ตัวแปร p_alive เป็นชนิดเลขจำนวนเต็ม และ กำหนดตัวแปร p_name เป็นชนิดตัวอักษรยาวไม่เกิน 10 ตัวอักษร ตัวแปรเหล่านี้จะรู้จักภายในการประมวลผลการทำซ้ำนี้เท่านั้น จากนั้นจะเรียกฟังก์ชันซีเอสพีอินเพื่อรับค่าจากโปรเซส sieve ต่าง ๆ (แบบไม่รอ) เพื่อเก็บค่าในตัวแปร n ถ้ามีการส่งค่ามาจากโปรเซส sieve ใด ๆ ฟังก์ชันนี้จะทำการพิมพ์ค่านั้นออกทางหน่วยแสดงผล แล้วกลับไปเริ่มประมวลผลการทำซ้ำรอบต่อไป แต่ถ้าไม่มีการส่งค่ามาจากโปรเซส sieve ใด ๆ จะหลุดจากการประมวลผลการทำซ้ำนี้

บรรทัดที่ 154 พิมพ์คำว่า job completed เพื่อแสดงการสิ้นสุดการทำงานออกทางหน่วยแสดงผล

บรรทัดที่ 155 จุดสิ้นสุดของฟังก์ชัน print

ข้อเสนอแนะ

ฟังก์ชันต่าง ๆ ที่พัฒนาขึ้นนี้มิได้ป้องกันการเกิดการอับจน (deadlock) (วิศวกรรมสถานแห่งประเทศไทย, 2527-2530) ซึ่งถ้าโปรแกรมประยุกต์มีการรอรับค่าเป็นวงกลม จะก่อให้เกิดการอับจนขึ้นได้ ดังนั้น หากมีการเพิ่มเติมไลบรารีฟังก์ชันเพื่อให้ป้องกันการเกิดการอับจน จะก่อให้เกิดประสิทธิผลยิ่งขึ้นฟังก์ชันทำงานได้มีประสิทธิภาพมากยิ่งขึ้น