CPA – Computació Paral·lela

Grau en Enginyeria Informàtica

S1. Introducció als Entorns de Programació Paral·lela

J. M. Alonso, P. Alonso, F. Alvarruiz, I. Blanquer,
J. Ibáñez, E. Ramos, J. E. Román

Departament de Sistemes Informàtics i Computació
Universitat Politècnica de València

Curs 2024/25

etsinf

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Contingut

Apartat 1

Programació en C

- Recordatori del Llenguatge C

El Llenguatge C

C és un llenguatge de programació de propòsit general

- Característiques: compilat, portable i eficient
- Java i C++ hereten la sintaxi de C
- Nucli del llenguatge simple, funcionalitat extra mitjançant biblioteques (llibreries)
- Un dels més utilitzats en supercomputació

```
void daxpy(int n, double a, double *x, double *y)
{
    int i;
    for (i=0; i<n; i++) {
        y[i] = a*x[i] + y[i];
    }
}
```

Variables i Tipus Bàsics

Totes les variables s'han de declarar

- Enters: `char`, `int`, `long`; modificador `unsigned`
- Enumerats: `enum` (equival a un enter)
- Coma flotant: `float`, `double`
- Tipus buit: `void` (ús especial)
- Tipus derivats: `struct`, arrays, punters

```
char c;
int i1,i2;
enum {NORD,SUD,EST,OEST} dir;
unsigned int k;
const float pi=3.141592;
double r=2.5,g;
```

```
c = 'M';
i1 = 2;
i2 = -5*i1;
dir = SUD;
k = (unsigned int) dir;
g = 2*pi*r;
```

Es poden definir nous tipus amb `typedef`

```
typedef enum {ROIG,VERD,BLAU,GROC,BLANC,NEGRE} color;
color c1,c2;
```


Sentències i Expressions

Existeixen diferents tipus de sentències:

- Declaració de variables i tipus (dins/fora de funció)
- Expressió, típicament una assignació `var=expr`
- Sentència composta (bloc `{...}`)
- Condicionals (`if`, `switch`), bucles (`for`, `while`, `do`)
- Unes altres: sentència buida (`;`), salt (`goto`)

Expressions:

- Assignacions: `=`, `+=`, `-=`, `*=`, `/=`; increments: `++`, `--`
- Aritmètiques: `+`, `-`, `*`, `/`, `%`; a nivell de bit: `~`, `&`, `|`, `^`, `<<`, `>>`
- Lògiques: `==`, `!=`, `<`, `>`, `<=`, `>=`, `||`, `&&`, `!`
  El zero s'assimila a "fals" i qualsevol altre a "vertader"
- Operador ternari: `a? b: c`

Exemples de Sentències de Control de Flux

```
if (j>0) valor = 1.0;
else valor = -1.0;
```

```
if (i>1 && (qi[i]-1.0)<1i-7) {
  zm1[i] *= 1.0+sk1[i-1];
  zm2[i] *= 1.0+sk1[i-1];
} else {
  zm1[i] *= 1.0+sk0[i-1];
  zm2[i] *= 1.0+sk0[i-1];
}
```

```
for (i=0;i<n;i++) x[i] = 0.0;
```

```
k = 0;
while (k<n) {
  if (a[k]<0.0) break;
  z[k] = 2.0*sqrt(a[k]);
  k++;
}
```

```
switch (dir) {
  case NORD:
    y += 1; break;
  case SUD:
    y -= 1; break;
  case EST:
    x += 1; break;
  case OEST:
    x -= 1; break;
}
```

```
for (i=0;i<n;i++) {
  y[i] = b[i];
  for (j=0;j<i;j++) {
    y[i] -= L[i][j]*y[j];
  }
  i[i] /= L[i][i] ;
}
```


Arrays i Punters

Array: col·lecció de variables del mateix tipus

- En la declaració s'indica la longitud
- Els elements s'accedeixen amb un índex (comença en 0)

```
#define N 10
int i;
double a[N],s=0.0;
for (i=0;i<N;i++)
  s = s + a[i];
```

Arrays multidimensionals: `double matriz[N][M];`
Les cadenes són arrays de `char` acabades amb caràcter `'\0'`

Punter: variable que conté l'adreça d'una altra variable

- En la declaració s'afig * abans del nom de variable
- L'operador & retorna l'adreça d'una variable
- L'operador * permet accedir a la dada apuntada

```
double a[4] =
 {1.1,2.2,3.3,4.4};
double *p,x;
p = &a[2];
x = *p;
*p = 0.0;
p = a;  /* &a[0] */
```

Més Sobre Punters

Aritmètica de punters

- Operacions bàsiques: +, −,++
- El desplaçament és del tipus al que apunta la variable

```
char s[] =
    "Comput. Paral·lela";
char *p = s;
while (*p!='P') p++;
```

Punter nul

- El seu valor és zero (NULL)
- S'usa per a indicar un error

```
double w,*p;
...
if (!p)
  error("Punter invàlid");
else w = *p;
```

Punter genèric

- De tipus void*
- Pot apuntar a variables de qualsevol tipus

```
void *p;
double x=10.0,z;
p = &x;
z = *(double*)p;
```

Punter múltiple: `double **p` (punter a punter)

Estructures

Estructura: col·lecció de dades heterogènies

- Els membres s'accedeixen amb . (o -> en el cas de punter a estructura)

```
struct vcomplex {
  double re,im;
};
struct vcomplex c1, *c2;
c1.re = 1.0;
c1.im = 2.0;
c2 = &c1;
c2->re = -1.0;
```

```
typedef struct {
  int i,j,k;
  const char *label;
  double data[100];
} mystruct;

mystruct s;
s.label = "NEW";
```

Funcions

Un programa C es compon d'almenys una funció (`main`)

Retornen un valor (llevat que la funció siga de tipus `void`)

```
double rad2deg(double x) {
  return x*57.29578;
}
```

```
void missatge(int k) {
  printf("Fi etapa %d\n",k);
}
```

Pas de paràmetres per valor (el pas per referència s'aconsegueix mitjançant punters)

```
float fun1(float a,float b){
  float c;
  c = (a+b)/2.0;
  return c;
}
...
w = fun1(6.0,6.5);
```

```
void fun2(float *a,float *b){
  float c;
  c = ((*a)+(*b))/2.0;
  if (fun3(c)*fun3(*a)<=0.0)
    *b = c;
  else *a = c;
}
...
fun2(&x,&i);
```

Es poden declarar funcions abans de la seua definició (prototip)

Funcions de Biblioteca

Operacions de cadenes `<string.h>`

- Copia cadena (`strcpy`), compara cadena (`strcmp`)
- Copia memòria (`memcpy`), inicialitza memòria (`memset`)

Entrada-eixida `<stdio.h>`

- Estàndard: `printf`, `scanf`
- Fitxers: `fopen`, `fclose`, `fprintf`, `fscanf`

Utilitats estàndard `<stdlib.h>`

- Gestió de memòria dinàmica: `malloc`, `free`
- Conversions: `atof`, `atoi`

Funcions matemàtiques `<math.h>`

- Funcions i operacions: `sin`, `cos`, `exp`, `log`, `pow`, `sqrt`
- Arrodoniment: `floor`, `ceil`, `fabs`

Exemple amb Fitxer

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

void lligdades( char *filename )
{
  FILE *fd;
  int i,n,*ia,*ja;
  double *va;
  fd = fopen(filename,"r");
  if (!fd) {
    perror("Error - fopen");
    exit(1);
  }
  fscanf(fd,"%i",&n);        /* nombre de dades a carregar */
  ia = (int*) malloc(n*sizeof(int));
  ja = (int*) malloc(n*sizeof(int));
  va = (double*) malloc(n*sizeof(double));
  for (i=0;i<n;i++) {
    fscanf(fd,"%i%i%lf",ia+i,ja+i,va+i);
  }
  fclose(fd);
  processa(n,ia,ja,va);
  free(ia); free(ja); free(va);
}
```


Tipus de Variables

Variables globals

- Es declaren fora de qualsevol funció
- Accés des de qualsevol punt del programa
- Es creen en el segment de dades

Variables locals

- Declarades dins d'una funció
- Visibles dins del bloc
- Es creen en la pila (*stack*), es destrueixen en eixir

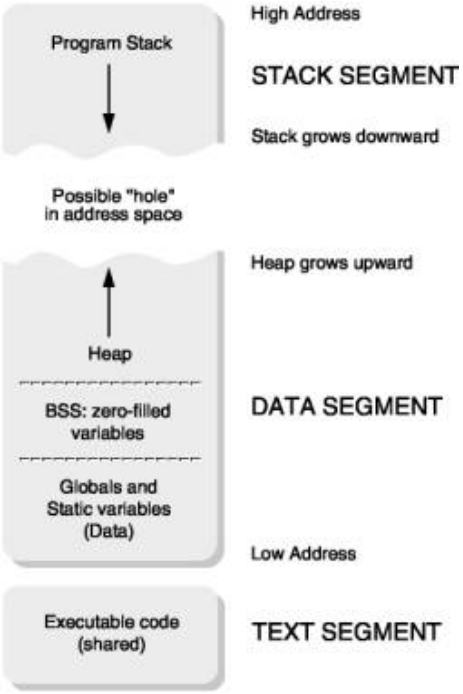
Variables estàtiques

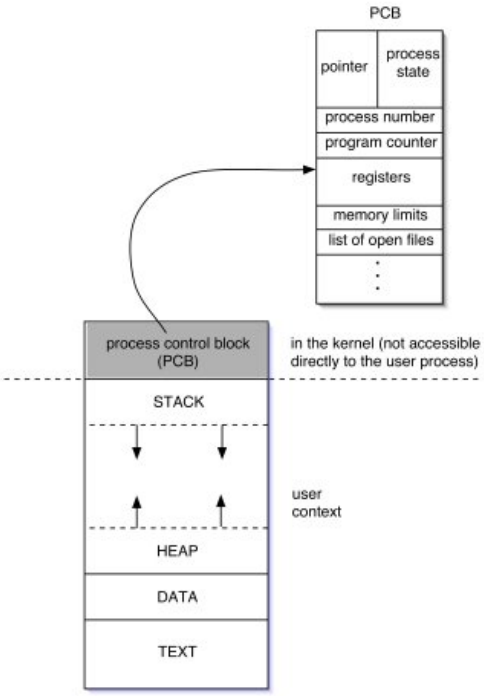
- Modificador `static`
- Àmbit local però persistents d'una crida a una altra

Variables en memòria dinàmica

- Memòria reservada amb `malloc`, persisteixen fins al `free`
- Es creen en el *heap*

Model de Memòria en Processos Unix



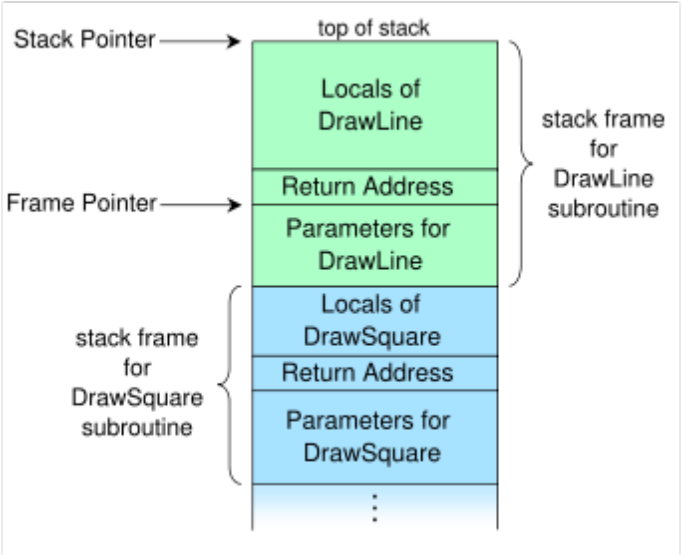


Pila de Crides

Els arguments d'una funció són com variables locals

En entrar en una funció:

1. s'apilen els arguments
2. s'apila l'adreça de tornada
3. es creen les variables locals



En fer `return` es destrueix tot el context creat

Apartat 2

Ús de Computadors Paral·lels

- Cicle de Desenvolupament

Cicle de Desenvolupament

El procés de compilació consta de:

- Preprocessat: modifica el codi font en C segons una sèrie d'instruccions (directives de preprocessat)
- Compilació: genera el codi objecte (binari) a partir del codi ja preprocessat
- Enllaçat: uneix els codis objecte dels diferents mòduls i biblioteques externes per a generar l'executable final

El cicle de desenvolupament es complementa amb altres passos:

- Automatitzar compilació de programes complexos (`make`)
- Depuració d'errors (`gdb`, `valgrind`)
- Anàlisi de prestacions (`gprof`)

Preprocessat

Com a pas previ a la compilació es fa el preprocessat (comando cpp, s'invoca automàticament)

- `include`: insereix el contingut d'un altre fitxer
- `define`: defineix constants i macros (amb arguments)
- `if`, `ifdef`: compilació condicional
- `pragma`: directiva de compilador

```
#include "myheader.h"

#define PI 3.141592
#define DEBUG_
#define AVG(a,b) ((a)+(b))/2

#ifdef DEBUG_
  printf("variable i=%d\n",i);
#endif
```


Compilació i Enllaçat

Compilació: `cc`

- Per cada fitxer `*.c` es genera un `*.o`
- Conté codi màquina de les funcions i variables, i una llista de símbols no resolts

Enllaçat o muntatge (*link*): `ld`

- Resol totes les dependències pendents a partir de `*.o` i biblioteques (`*.a`, `*.so`)

ex.c

```
#include <stdio.h>
extern double f1(double);
int main() {
  double x = f1(4.5);
  printf("x = %g\n",x);
  return 0;
}
```

f1.c

```
#include <math.h>
double f1(double x) {
  return 2.0/(1.0+log(x));
}
```

```
$ gcc -o ex ex.c f1.c -lm
```

Compilació de Programes Paral·lels

OpenMP es basa en directives `#pragma omp`

- Un compilador sense suport OpenMP ignora aquestes directives
- Els compiladors recents tenen suport, amb una opció (és necessària tant al compilar como al enllaçar)

```
$ gcc -fopenmp -o prgomp prgomp.c
```

MPI proporciona el comando `mpicc`

- Invoca a `cc` afegint totes les opcions necessàries (biblioteques de MPI, ruta de `mpi.h`)
- Facilita la compilació en diferents màquines
- `mpicc -show` mostra les opcions que s'usaran
- També `mpicxx`, `mpif77`, `mpif90`

```
$ mpicc -o prgmpi prgmpi.c
```