

Make

Nikola Brković

15. januar 2025

Kaj je Make?

- Orodje, ki avtomatizira proces izgradnje programske opreme
- Omogoča opis odvisnosti med izvornimi datotekami v konfiguracijski datoteki - Makefile

- GNU, BSD, NMAKE, dmake
- Danes se bomo osredotočili na GNU Make
- Podpira Linux, Mac OS, Windows, OS/2, DOS..

- Vsebuje seznam ciljev
- Cilj je odvisen od predpogojev in lahko vsebuje pravila
- Podpira tudi spremenljivke, pogojno izvajanje, itd.

```
cilj [cilj ...]: [predpogoj ...]  
[prvo pravilo]  
...  
[zadnje pravilo]
```

Primer preprostega Makefila

```
all: libfile.a main.o
    ld libfile.a main.o

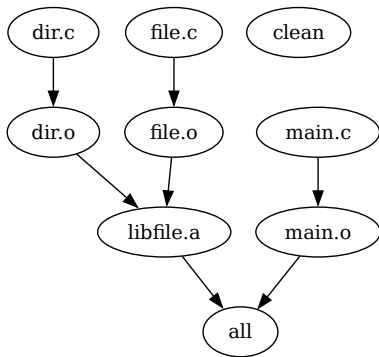
main.o: main.c
    gcc -c main.c -o main.o

dir.o: dir.c
    gcc -c dir.c -o dir.o

file.o: file.c
    gcc -c file.c -o file.o

libfile.a: dir.o file.o
    ar rcs libfile.a dir.o file.o
```

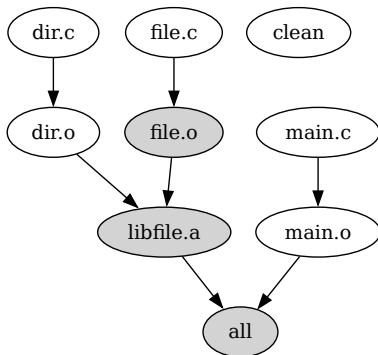
Graf odvisnosti



- 1 Če je začetni cilj umeten, in nima nobenih predpogojev, se bo izvedel in se postopek konča. (*clean*)
- 2 Spustimo se po grafu odvisnosti, od vključno trenutnega cilja, v nasprotni smeri povezav, dokler ne naletimo na cilj, katerega čas zadnje spremembe je starejši kot čas zadnje spremembe enega od njegovih predpogojev.
- 3 Če ni nobenega zastarelega cilja v celotnem grafu, se postopek konča, sicer pa najdeni cilj izvedemo.
- 4 Vrnemo se v prejšnji cilj, poskušamo ponoviti korak 2 čim večkrat. Preskočimo cilje, ki smo jih morebiti že izvedli. Če zastarelih ciljev ne najdemo več, izvedemo trenutni cilj in ponovimo ta korak.
- 5 Končamo, ko smo prišli nazaj v začetni cilj.

Izvajanje ciljev

Spremenili smo datoteko file.c:



Zaporedno izvajanje

- Izvajanje več poslov hkrati
- Poslovne reže (ang. *job slots*)

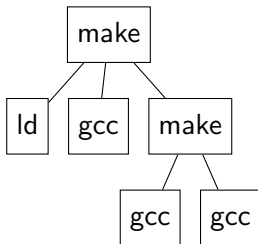
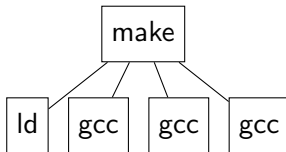
```
make -j 4
```

- Ključno pri velikih projektih
- Primer: Linux (linux-6.13-rc7)

```
$ find . -name '*.c' | wc -l  
34788
```

Zaporedno izvajanje

Pri enem procesu je preprosto, ampak na težave naletimo pri rekurzivnih Makih, kjer je potrebna medprocesna komunikacija. Rešeno z jobserverjem.



Implementiran z uporabo poimenovane cevi (FIFO), imamo N poslovnih rež.

Ko želi zagnati posel, mora zasesti režo. Ko se posel konča, pa sprosti režo.

① Začetno stanje

→

3	7	2
---	---	---

 →

② Reža zasedena, prebran žeton

→

3	7
---	---

 →

③ Zapisan žeton

→

2	3	7
---	---	---

 →

Na starejših sistemih na režo čaka z *read*, na novejših pa *pselect*.

```
static int job_rfd = -1;

static int
make_job_rfd ()
{
    EINTRLOOP (job_rfd , dup (job_fds[0]));
    // ...
    return job_rfd;
}
```

```
void
jobserver_signal ()
{
    if (job_rfd >= 0)
    {
        close (job_rfd);
        job_rfd = -1;
    }
}
```

unsigned int

jobserver_acquire (**int** timeout)

{

// ...

EINTRLOOP (got_token , read (job_rfd , &intake , 1))

// ...

}

Implementiran z uporabo poimenovanega semaforja - systemskega atomskega števca.

Semafor se ustvari s funkcijo *CreateSemaphore*.

Lahko čakamo na proces in semafor hkrati z *WaitForMultipleObjects*.

Jobserver na Windowsih

```
HANDLE *handles;  
DWORD dwHandleCount;  
DWORD dwEvent;  
  
handles = xmalloc(process_table_actual_size()  
                  * sizeof(HANDLE));  
  
/* Add jobserver semaphore to first slot. */  
handles[0] = jobserver_semaphore;  
  
/* Build array of handles to wait for. */  
dwHandleCount = 1 + process_set_handles (&handles[1]);  
  
dwEvent = process_wait_for_multiple_objects (  
    dwHandleCount, /* number of objects in array */  
    handles,       /* array of objects */  
    FALSE,         /* wait for any object */  
    INFINITE);     /* wait until object is signalled */
```