Google Summer of Code Project

Parallelization in The FreeBSD Ports Collection

student:
Alexander Pronin
apronin@me.com

mentor:
Marcus von Appen

Contents

Problem

1. Safe way to build/install several ports at the same time
2. Convenient approach for parallel port dependency builds
3. Redesign make output

Things to consider

- Maintain the system in consistent state
- Prevent concurrent access to shared directories and files
- Avoid deadlock situations
- Failures handling

Contents

General development approach

- Use «clean room» approach
- Do not flush **bsd.port.mk** file
- **bsd.parallel.mk** concentrates parallel related targets, global variables, algorithms
- **${_parv_WANT_PARALLEL_BUILD}** - triggers parallel execution of build/install process

Example:

```
.if defined(WANT_PARALLEL_BUILD)
.include "${PORTSDIR}/Mk/bsd.parallel.mk"
.endif
```

General development approach

- dependency builds are background **sh(1)** processes

Example:

```
( \
    cd $$dir; ${MAKE} -DINSTALLS_DEPENDS $$target $$depends_args \
) & spawned=$$!; \
```

General development approach

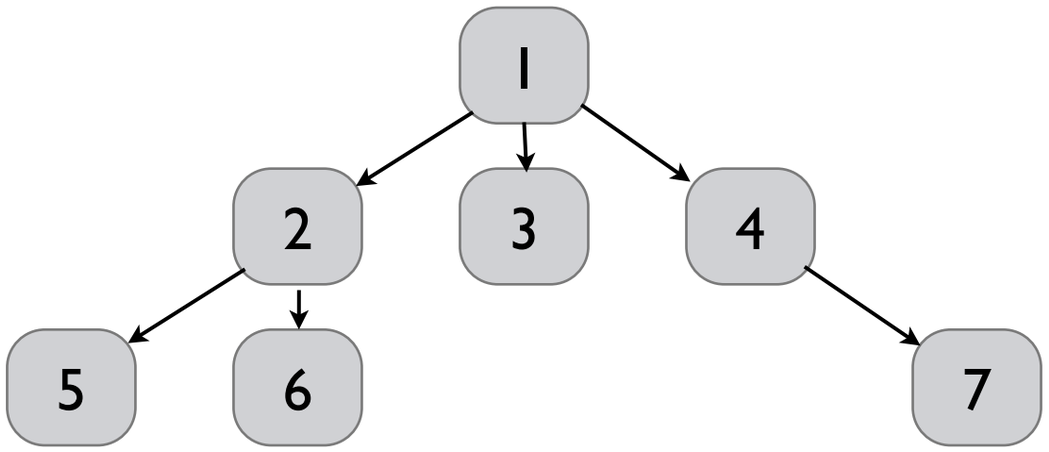
Termination of process tree

- **Soft termination.** Parent make process will not respond to "soft" kill signals, until all background dependency builds will be processed.
- **Hard termination**. Keyboard interrupts and hard kill signal (SIGKILL) kills parent make process, but this action leaves background child processes alive. These processes do not have assign terminal, moreover they will waste CPU time. Hence, it will be a challenge for the user to terminate all this processes.

General development approach

Termination of process tree

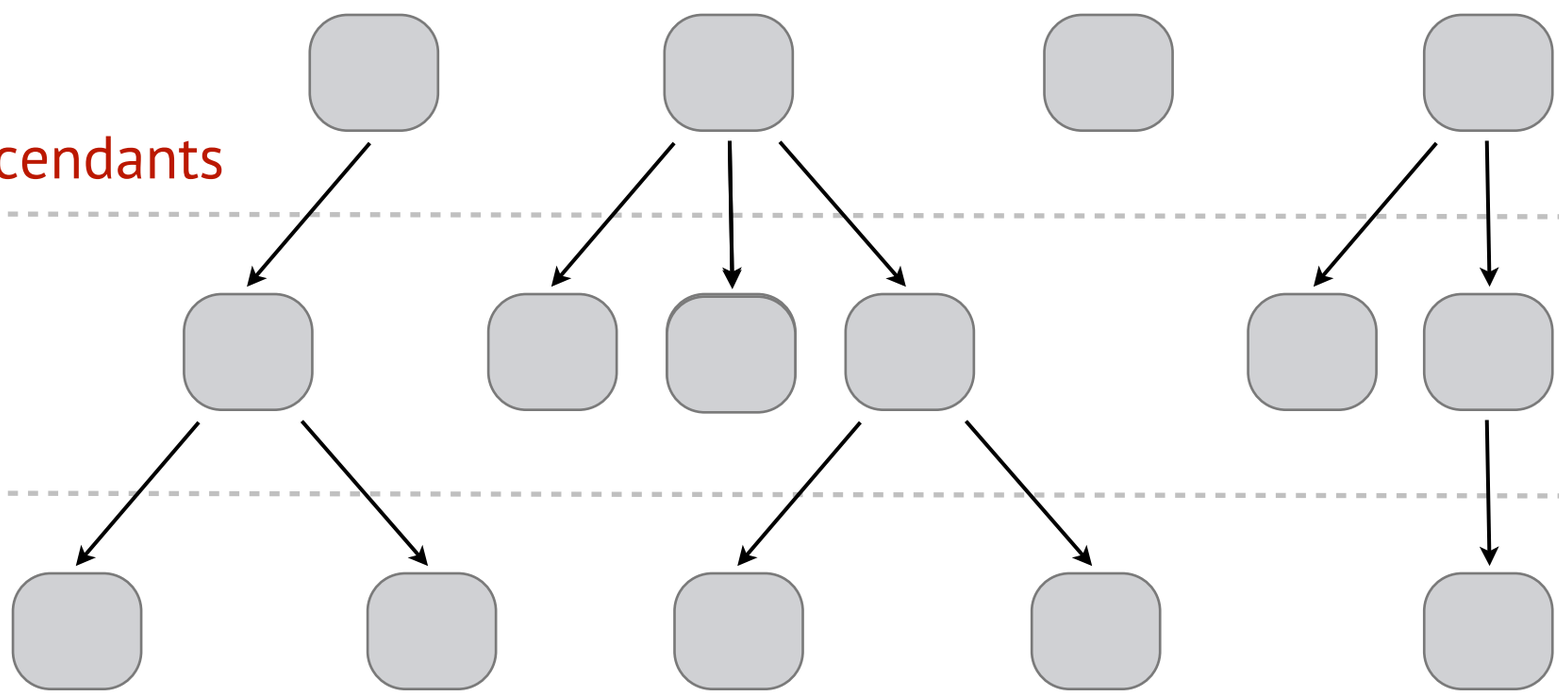
- Use **sh(1)** built-in command **trap** to listen to specific signals and act accordingly.
- Distinguish parent make process from dependency builds, using **${INSTALLS_DEPENDS}** global variable
- Signals to catch: **EXIT INT TERM**
- Explore process tree using **Breadth-first** search (BFS)



General development approach

Termination of process tree

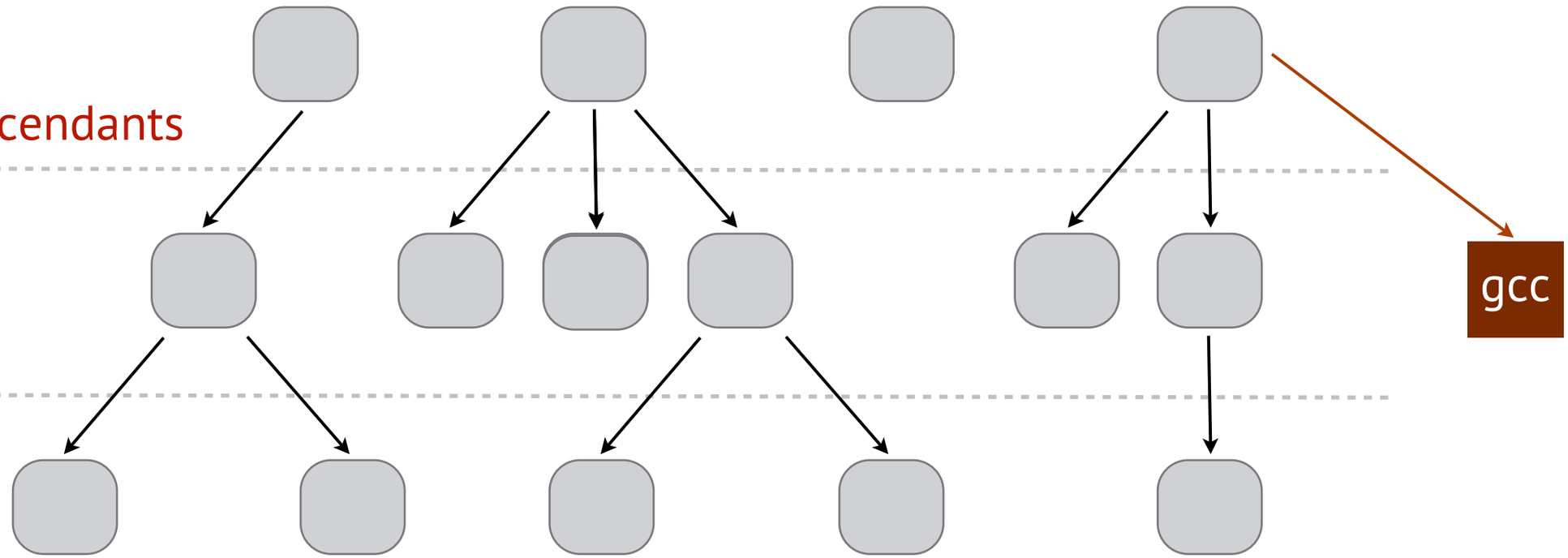
- send SIGSTOP
- store PIDs
- find direct descendants



General development approach

Termination of process tree

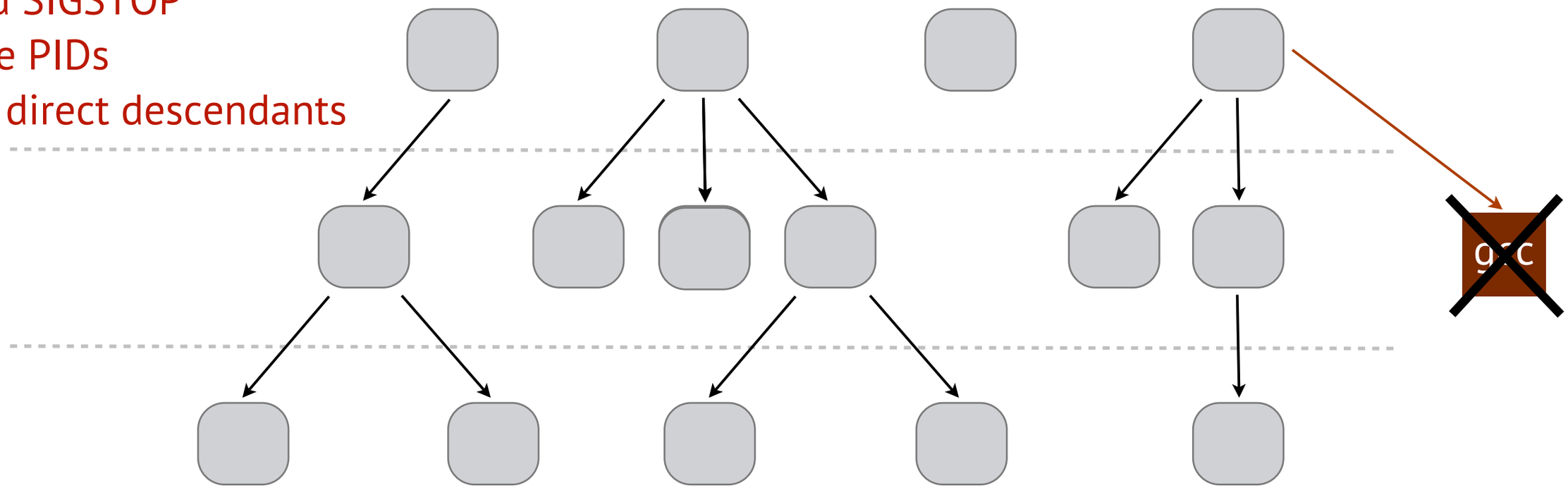
- send SIGSTOP
- store PIDs
- find direct descendants



General development approach

Termination of process tree

send SIGSTOP
store PIDs
find direct descendants



General development approach

"Infinity" loops

Ports collection lacks:

- wait/notify approach
- process manager
- multithreading infrastructure

- Port is responsible for granting CPU time to other processes
- Sleep timeouts are user configurable

Example:

```
>cat /etc/make.conf
# _parv_CHECK_ACTIVE_TIMEOUT - timeout in seconds before next check of active builds
#                              in case if port is prohibit to spawn another background
#                              process. Default: 2
_parv_CHECK_ACTIVE_TIMEOUT= 5
```

Contents

Parallel installation of several ports

Locking technique

- **LOCK** file approach prevents concurrent access problems
- The first process that placed **LOCK** file gets exclusive access to shared data
- **Subproblem:** consider stalled locks, e.g. if some process exits unexpectedly and leaves LOCK file.
- **Solution:** place lockers PID to lock file

Example:

```
...
pid=$$(${CAT} ${${lock_dir}}/${lock_file}); \
if [ $${pid} ]; then \
        ps -p $${pid} > /dev/null && status=$$? || status=$$?; \
....
```

Parallel installation of several ports

Locking technique

- Sequence of commands to lock a directory must be atomic
- No other process could be able to make attempts to lock a directory, while another process has been trying to lock the same directory.
- **lockf(1)** allows to execute a command while holding an exclusive file lock

Example:

```
lockf  ${lock_file} ${SH} -c '${LOCK_SEQ}'
```

Parallel installation of several ports

Port's directory locking

- **${WRKDIR}** is inappropriate for **LOCK**. Firstly, it is created only during extract phase. Secondly, user may change.
- **${LOCK_DIR}** global variable specifies a directory that collects ports **LOCK** files for the Ports Collection.
- **${PKGNAME}** is used for port's **LOCK** file naming.
- **Locking phases:** it is necessary to lock port as soon as possible and unlock it after all stuff is done. **.BEGIN** and **.END** targets are sufficient for this purpose.

Parallel installation of several ports

${PKG_DBDIR} locking

- A directory, where package installation is recorded
- Default: /var/db/pkg
- Does not changed by user

Locking phases:

1. During package registration.

fake-pkg target contains all magic related to package registration.

Example:

```
_INSTALL_SUSEQ= check-umask install-mtree pre-su-install
    ...
    install-ldconfig-file lock-pkg-dbdir fake-pkg unlock-pkg-dbdir security-check
    ...
```

Parallel installation of several ports

${PKG_DBDIR} locking

Locking phases:

2. During dependency installation.

XXX-depends tragets (pkg-depends, extract-depends, patch-depends, lib-depends, fetch-depends, build-depends, run-depends) are responsible for installation of port's dependencies.

${PKG_DBDIR} locking phase is reduced to package registration checking, which provides more access to shared directory to other processes.

Parallel installation of several ports

${PORT_DBDIR} locking

- A directory where port configuration options are recorded.
- Default: /var/db/ports
- Does not changed by user

Locking phase:

${PORT_DBDIR} is locked during whole stage of OPTIONS configuration for a specific port (make config)

Parallel installation of several ports

Locking technique

XXX-depends targets have to consider current installation of dependency ports.

Important: if a port determines that one of its dependencies is being built by another port, it does not mean that this dependency is processed.

Parallel installation of several ports

Redesign of conflicts checking

Sufficient sequence of conflits checks does not requare any locks:

1.Check conflicts against currently installing ports

2.Check conflicts against ports that are registered in **${PKG_DBDIR}**

Example:

check-conflicts: **check-active-build-conflicts** check-build-conflicts check-install-

Provide user with a **talkative** feedback, so that it will be obvious what is going on:

Example:

```
${ECHO_MSG} "===> ${PKGNAME} conflicts with currently installing package(s): "; \
for entry in $${conflicts_with}; do ${ECHO_MSG} "          $${entry}";  done; \
${ECHO_MSG} "     Please remove them first with pkg_delete(1)."; \
exit 1; \
```

Contents

Parallel installation of ports dependencies

Redesign of dependency builds

Default dependency checking algorithm:

- Check whether some dependency port installed or not.
- If dependency port is not installed then installation of dependency port starts. Naturally this blocks execution flow until this dependency will be processed.
- If dependency port is locked by another make process then the execution flow will be blocked, either.

This kills parallel benefits.

Parallel installation of ports dependencies

Track processing of spawned dependencies

- **depends** - A list of "path:dir[:target]" tuples of other ports this package depends on, which are not processed.
- **active_builds** - A list of "spawned_pid:path:dir[:target]" tuples of currently building dependencies.

sh(1) builtin **job control** allows tracking evaluation and exit codes of background processes.

Parallel installation of ports dependencies

OPTIONS and INTERACTIVE targets

- Every dependency blocks execution when waiting for user input.
- Process is unable to interact with user, if it was spawned as background process.

Requirement: Process all stages that require user intervention

- as soon as possible
- before parallelization
- in non-parallelized manner

run-depends-list and **build-depends-list** targets retrieves dependency ports for recursive OPTIONS processing.

Parallel installation of ports dependencies

Degree of parallelization

${_parv_PARALLEL_BUILDS_NUMBER} - global variable that controles number of parallel builds.

Both **default** and **maximum** number of parallel dependency builds for current port.

_parv_DEFAULT_PAR_BUILDS_NUM!= ${SYSCTL} -n kern.smp.cpus

Parallel installation of ports dependencies

Redesign of make output

Requirements:

- User must not get a sence of deadlock
- Does not annoy user with noise on console

Use user configurable feedback timeout (define specific global variable in **/etc/make.conf** file)

Important feedback information:

- What directory has to be locked now
- What dependency has just been spawned
- List of currently building dependencies
- Why make process is unable to spawn more dependencies

Parallel installation of ports dependencies

Redesign of make output

Redirect dependency output to different file and inform user about it.
Use **tail -F file_name** or similar utility to track processing of dependency if necessary.

Error feedback:

Example:

```
${ECHO_CMD} "Errors occured while building a dependency port \
        $$(cd $${dir}; ${MAKE} -V PKGNAME"; \
${ECHO_CMD} "Checkout its log"; \
${ECHO_CMD} "    ${_parv_PORTS_LOGS_DIR}/${_parv_PORT_LOG_FILE})"; \
${ECHO_CMD} "Terminating..."; \
exit 1; \
```

EuroBSDcon 2012 | Warsaw, Poland

Alexander Pronin

Parallelization in The FreeBSD Ports Collection

GitHub: https://github.com/scher/portsCollection

Wiki page: http://wiki.freebsd.org/SummerOfCode2012/Parallelization_in_the_ports_collection

E-mail: apronin@me.com