MyFind Protokoll

Team: *Oishik Roy*

Darko Djolev

Einführung

Dieses Dokument skizziert die Parallelisierungskonzepte und Synchronisationsmechanismen, die im "myfind"-Programm implementiert sind. Das Ziel des Programms ist es, parallel nach bestimmten Dateien innerhalb einer Verzeichnisstruktur zu suchen, wobei geforkte Child-Prozesse verwendet werden. Zusätzlich unterstützt das Programm eine Option für case-insensitive und rekursive Suchen.

Parallelisierungsansätze

1. ### Forking von Child-Prozessen

   Die primäre Parallelisierungstechnik beinhaltet die Verwendung des fork-Systemaufrufs, um Child-Prozesse zu erstellen. Jeder Child-Prozess ist dafür verantwortlich, nach einer bestimmten Datei im angegebenen Suchpfad zu suchen. Die Duplizierung des Parent-Prozesses ermöglicht die gleichzeitige Ausführung von Dateisuchen.

```c
// fork a new process
pid_t child_pid = fork();

// check if fork was successful
if (child_pid == -1)
{
    perror("fork");
    exit(EXIT_FAILURE);
}

// check if process is the child
if (child_pid == 0)
{
    // in the child process, search for files in the directory
    searchDirectory(searchPath, fileName, recursive, caseInsensitive, &foundInCurrentProcess);
    exit(foundInCurrentProcess);
```

2. ### Rekursive Suche

   Die `-R` Option ermöglicht eine rekursive Dateisuche. Wenn der aktuelle Verzeichniseintrag ein Unterverzeichnis darstellt (`S_ISDIR(statbuf.st_mode)` ist wahr) und die Rekursion aktiviert ist (`recursive` ist wahr), wird die `searchDirectory`-Funktion rekursiv aufgerufen, diesmal für das durch `fullPath` angegebene Unterverzeichnis.

```c
// recursive logic
if (S_ISDIR(statbuf.st_mode) && recursive)
{
    searchDirectory(fullPath, fileName, recursive, caseInsensitive, fileFound);
}
else
{
    // search for the file in the current directory
    searchFile(fullPath, fileName, caseInsensitive, fileFound);
}
```

3. ### Case-Insensitive Suche

Das Programm unterstützt sowohl case-sensitive als auch case-insensitive Dateisuchen. Wenn die `-i` Option verwendet wird, führt das Programm case-insensitive Suchen durch, indem es die `strcasecmp`-Funktion verwendet, um Dateinamen ohne Berücksichtigung der Groß-/Kleinschreibung zu vergleichen. Ohne die `-i` Option führt das Programm standardmäßig case-sensitive Suchen durch und verwendet die `strcmp`-Funktion, um Dateinamen case-sensitive zu vergleichen.

```
// case insensitive logic
if (S_ISREG(statbuf.st_mode) &&
    ((caseInsensitive && strcasecmp(fileName, strrchr(fullPath, '/') + 1) == 0) ||
     (!caseInsensitive && strcmp(fileName, strrchr(fullPath, '/') + 1) == 0)))
{
    printf("%d: %s: %s\n", getpid(), fileName, fullPath);
    *fileFound = 1;
}
```

4. ### Warten auf Child-Prozesse

Der Parent-Prozess wartet auf die Beendigung aller Child-Prozesse mittels des `wait`-Systemaufrufs. Dies stellt sicher, dass der Parent nicht fortfährt, bevor alle Dateisuchen abgeschlossen sind. Nachdem eine Datei gefunden wurde, wird der Prozess beendet und gibt die Nachricht „Prozess ID xxx was terminated“ aus. Wenn keine Datei im Verzeichnis gefunden wurde, erscheint die Meldung „No files in xxx found.“.

```
        // in the parent process, wait for the child process to complete
        int status;
        waitpid(child_pid, &status, 0);

        // check if files were found in the child process
        if (WIFEXITED(status) && WEXITSTATUS(status) != 0)
        {
            fileFound = 1;
            printf("Process ID %d was terminated.\n", child_pid);
        }
        else if (WIFEXITED(status) && WEXITSTATUS(status) == 0 && fileFound == 0)
        {
            fileFound = 0;
        }
    }
}

// output if no files were found
if (!fileFound)
{
    printf("No files in %s found\n", searchPath);
}
```

Die implementierten Parallelisierungskonzepte nutzen geforkte Child-Prozesse, um gleichzeitige Dateisuchen zu ermöglichen. Der Synchronisationsmechanismus sorgt für eine ordnungsgemäße Koordination zwischen Parent- und Child-Prozessen, verhindert Race-Conditions und garantiert den Abschluss der Suchaufgaben.