ECE30021/ITP30002 Operating System

Programming Assignment 3

# Dynamic Deadlock Detection & Prediction

# Overview

- Develop a runtime monitoring tool using runtime inter-positioning to detects & predicts resource deadlocks
  - Target to detect/predict cyclic deadlocks in multithreaded C programs using Pthread
  - Construct two dynamic libraries to override Pthread
    - online deadlock detector
    - offline deadlock predictor
- Demonstrate your implementation effectively detects and predicts deadlock cases
  - Use the given example programs
  - Construct more false-positive examples

# Background: Runtime Interpositioning

- If LD_PRELOAD environment variable is set to a list of shared library path, then when you load and execute a program, the dynamic linker will search the LD_PRELOAD libraries before any other shared libraries

- Examples
  - random examples
  - malloc example

- addr2line

  $ addr2line <addr> -e <executable>

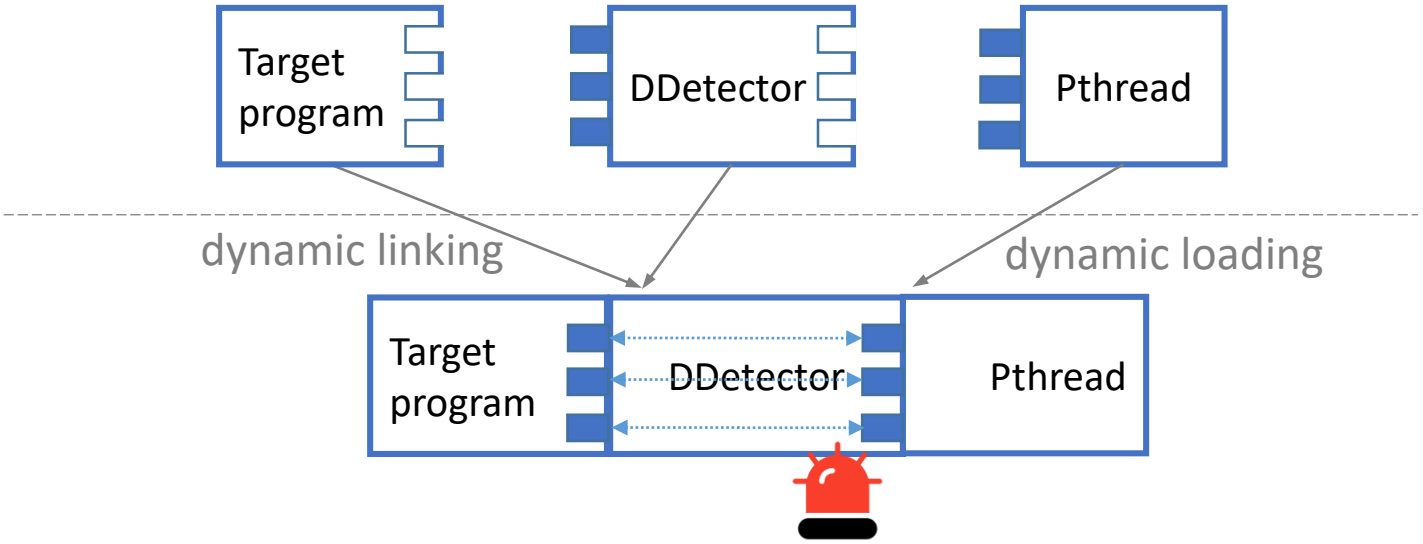
# Tasks

- Task 1. Online deadlock detector
  - alert a user to the occurrence of a cyclic deadlock at a target program execution

- Task 2. Offline deadlock predictor
  - extract a runtime trace of a target program execution
  - run an analysis on the runtime trace to check potential deadlocks

# Task 1. Deadlock Detector

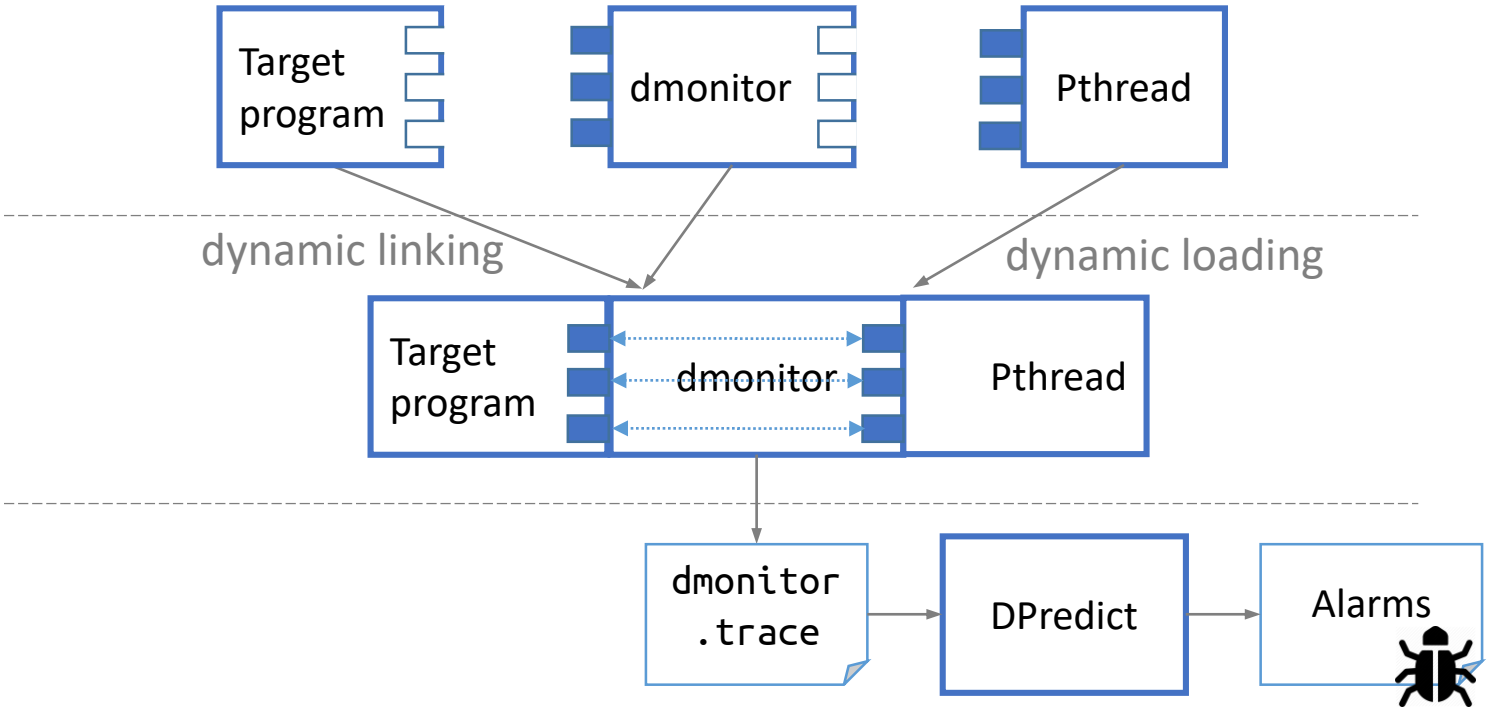
- Construct a cyclic deadlock detector *DDetector*
  - Implement the algorithm described in Page 8 of "Deadlock Detection"
    - Construct DDetector as a dynamic library `ddetector.so` (`ddetector.c`) that overrides certain Pthread API functions
  - Online analysis: DDetector prints out an alert message to standard error in runtime when the target program execution falls into a cyclic deadlock
    - must be side-effect-free (i.e., do not change target program behaviors)



# Task 2. Deadlock Predictor (1/3)

- Construct a cyclic deadlock predictor *DPredictor*
  - Implement the Goodloock algorithm described in "Deadlock Detection"
    1. dynamic library `dmonitor.so (dmonitor.c)`
    2. offline trace analyzer `dpredictor` (`dpredictor.c`)
  - Workflow
    - `dmonitor` is dynamically linked with the target program
    - `dmonitor` extracts the runtime trace of a target program execution for deadlock prediction
      - generate `dmonitor.trace`
    - after the target program execution, `dpredictor` reads `dmonitor.trace` and detects potential cyclic deadlocks

# Task 2. Deadlock Predictor (2/2)




- A potential deadlock must be reported with the following details
  - The number of threads associated with the potential deadlock
  - Source code locations of the lock operations associated with the potential deadlocks

# Requirements

- Assumptions
  - A target program creates no more than 10 threads
  - A target program creates no more than 100 mutexes
- Your write-up represents how DDetector and DPredictor are built to implement the corresponding algorithms in detail
  - e.g., describe which Pthread APIs are overridden and why
- You must write example programs to show the two checkers accurately detects deadlocks
  - especially, different false-positive cases for DPredictor
- Submit the source code files as well as the build scripts for the techniques and your example programs
- Take a video clip of the demonstration
  - less than 5 minutes

# Submission

- Deadline: 11:59 PM, 31 May (Fri)
  - late submission will be accepted in the next 24 hours with 20% penalty

- Your submission must include the followings:
  - Write-up: up to 5 pages (either in single- or double-columns)
  - URL of your video demo (e.g., YouTube)
    - put the URL in your write-up
  - All related source code files

- How to submit
  - upload your files to a homework repository in Hisnet
  - by only one of the team member

# Evaluation

- Points
  - Technical soundness 40%
  - Demonstration 30%
  - Presentation 20%
  - Discussion 10%

- Note
  - Evaluation will be primary on your write-up and video demo
  - TAs will test the submitted files on the peace server