Subject: Re: [PATCH] ltp controllers: block device i/o bandwidth controller testcase (was: Re: [LTP] [PATCH 0
Posted by Subrata Modak on Mon, 07 Jul 2008 10:24:37 GMT
View Forum Message <> Reply to Message

Thanks Andrea for contributing these tests to LTP. I will take some time
before testing this and coming back to you. Give me some time. Meanwhile
it would be great if Balbir/Dhaval/others can provide some review
comments as well.

Regards--
Subrata

On Mon, 2008-07-07 at 12:05 +0200, Andrea Righi wrote:
> Add the block device I/O bandwidth controller (io-throttle) testcase.
>
> See testcase documentation for design and implementation details.
> See also: http://lkml.org/lkml/2008/7/4/143
>
> Signed-off-by: Andrea Righi <righi.andrea@gmail.com>
> ---
> diff --exclude CVS -urpN ltp/testcases/kernel/controllers.orig/io-throttle/iobw.c
ltp/testcases/kernel/controllers/io-throttle/iobw.c
> --- ltp/testcases/kernel/controllers.orig/io-throttle/iobw.c 1970-01-01 01:00:00.000000000 +0100
> +++ ltp/testcases/kernel/controllers/io-throttle/iobw.c 2008-07-07 11:44:51.000000000 +0200
> @@ -0,0 +1,279 @@
> +/*
> + * iobw.c - simple I/O bandwidth benchmark
> + *
> + * This program is free software; you can redistribute it and/or
> + * modify it under the terms of the GNU General Public
> + * License as published by the Free Software Foundation; either
> + * version 2 of the License, or (at your option) any later version.
> + *
> + * This program is distributed in the hope that it will be useful,
> + * but WITHOUT ANY WARRANTY; without even the implied warranty of
> + * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
> + * General Public License for more details.
> + *
> + * You should have received a copy of the GNU General Public
> + * License along with this program; if not, write to the
> + * Free Software Foundation, Inc., 59 Temple Place - Suite 330,
> + * Boston, MA 021110-1307, USA.
> + *
> + * Copyright (C) 2008 Andrea Righi <righi.andrea@gmail.com>
> + */
> +
> +#define _GNU_SOURCE

```
> +#define __USE_GNU
> +
> +#include <errno.h>
> +#include <stdio.h>
> +#include <stdlib.h>
> +#include <malloc.h>
> +#include <fcntl.h>
> +#include <signal.h>
> +#include <string.h>
> +#include <unistd.h>
> +#include <sys/types.h>
> +#include <sys/stat.h>
> +#include <sys/time.h>
> +#include <sys/wait.h>
> +#include <asm/page.h>
> +
> +#define ALIGN(x,a)  __ALIGN_MASK(x,(typeof(x))(a)-1)
> +#define __ALIGN_MASK(x,mask) (((x)+(mask))&~(mask))
> +#define KB(x)   ((x) >> 10)
> +
> +const char usage[] = "Usage: iobw [-direct] threads chunk_size data_size\n";
> +const char child_fmt[] =
> +  "(%s) task %3d: time %4lu.%03lu bw %7lu KiB/s (%s)\n";
> +const char parent_fmt[] =
> +  "(%s) parent %d: time %4lu.%03lu bw %7lu KiB/s (%s)\n";
> +
> +static int directio = 0;
> +static size_t data_size = 0;
> +static size_t chunk_size = 0;
> +
> +typedef enum {
> + OP_WRITE,
> + OP_READ,
> + NUM_IOPS,
> +} iops_t;
> +
> +static const char *iops[] = {
> + "WRITE",
> + "READ ",
> + "TOTAL",
> +};
> +
> +static int threads;
> +pid_t *children;
> +
> +char *mygroup;
> +
> +static void print_results(int id, iops_t op, size_t bytes, struct timeval *diff)
```

```
> +{
> + fprintf(stdout, id ? child_fmt : parent_fmt,
> +  mygroup, id, diff->tv_sec, diff->tv_usec / 1000,
> +  (bytes / (diff->tv_sec * 1000000L + diff->tv_usec))
> +  * 1000000L / 1024, iops[op]);
> +}
> +
> +static void thread(int id)
> +{
> + struct timeval start, stop, diff;
> + int fd, i, ret;
> + size_t n;
> + void *buf;
> + int flags = O_CREAT | O_RDWR | O_LARGEFILE;
> + char filename[32];
> +
> + ret = posix_memalign(&buf, PAGE_SIZE, chunk_size);
> + if (ret < 0) {
> +  fprintf(stderr,
> +   "ERROR: task %d couldn't allocate %lu bytes (%s)\n",
> +   id, chunk_size, strerror(errno));
> +  exit(1);
> + }
> + memset(buf, 0xaa, chunk_size);
> +
> + snprintf(filename, sizeof(filename), "%s-%d-iobw.tmp", mygroup, id);
> + if (directio)
> +  flags |= O_DIRECT;
> + fd = open(filename, flags, 0600);
> + if (fd < 0) {
> +  fprintf(stderr, "ERROR: task %d couldn't open %s (%s)\n",
> +   id, filename, strerror(errno));
> +  free(buf);
> +  exit(1);
> + }
> +
> + /* Write */
> + lseek(fd, 0, SEEK_SET);
> + n = 0;
> + gettimeofday(&start, NULL);
> + while (n < data_size) {
> +  i = write(fd, buf, chunk_size);
> +  if (i < 0) {
> +   fprintf(stderr, "ERROR: task %d writing to %s (%s)\n",
> +    id, filename, strerror(errno));
> +   ret = 1;
> +   goto out;
> +  }
```

```
> +  n += i;
> + }
> +        gettimeofday(&stop, NULL);
> +        timersub(&stop, &start, &diff);
> + print_results(id + 1, OP_WRITE, data_size, &diff);
> +
> + /* Read */
> + lseek(fd, 0, SEEK_SET);
> + n = 0;
> + gettimeofday(&start, NULL);
> + while (n < data_size) {
> +  i = read(fd, buf, chunk_size);
> +  if (i < 0) {
> +   fprintf(stderr, "ERROR: task %d reading to %s (%s)\n",
> +    id, filename, strerror(errno));
> +   ret = 1;
> +   goto out;
> +  }
> +  n += i;
> + }
> +        gettimeofday(&stop, NULL);
> +        timersub(&stop, &start, &diff);
> + print_results(id + 1, OP_READ, data_size, &diff);
> +out:
> + close(fd);
> + unlink(filename);
> + free(buf);
> + exit(ret);
> +}
> +
> +static void spawn(int id)
> +{
> + pid_t pid;
> +
> + pid = fork();
> + switch (pid) {
> + case -1:
> +  fprintf(stderr, "ERROR: couldn't fork thread %d\n", id);
> +  exit(1);
> + case 0:
> +  thread(id);
> + default:
> +  children[id] = pid;
> + }
> +}
> +
> +void signal_handler(int sig)
> +{
```

```
> + char filename[32];
> + int i;
> +
> + for (i = 0; i < threads; i++)
> +  if (children[i])
> +   kill(children[i], SIGKILL);
> +
> + for (i = 0; i < threads; i++) {
> +  struct stat mystat;
> +
> +  snprintf(filename, sizeof(filename), "%s-%d-iobw.tmp",
> +    mygroup,i);
> +  if (stat(filename, &mystat) < 0)
> +   continue;
> +  unlink(filename);
> + }
> +
> + fprintf(stdout, "received signal %d, exiting\n", sig);
> + exit(0);
> +}
> +
> +unsigned long long memparse(char *ptr, char **retptr)
> +{
> + unsigned long long ret = strtoull(ptr, retptr, 0);
> +
> + switch (**retptr) {
> + case 'G':
> + case 'g':
> +  ret <<= 10;
> + case 'M':
> + case 'm':
> +  ret <<= 10;
> + case 'K':
> + case 'k':
> +  ret <<= 10;
> +  (*retptr)++;
> + default:
> +  break;
> + }
> + return ret;
> +}
> +
> +int main(int argc, char *argv[])
> +{
> + struct timeval start, stop, diff;
> + char *end;
> + int i;
> +
```

```
> + if (argv[1] && strcmp(argv[1], "-direct") == 0) {
> +  directio = 1;
> +  argc--;
> +  argv++;
> + }
> + if (argc != 4) {
> +  fprintf(stderr, usage);
> +  exit(1);
> + }
> + if ((threads = atoi(argv[1])) == 0) {
> +  fprintf(stderr, usage);
> +  exit(1);
> + }
> + chunk_size = ALIGN(memparse(argv[2], &end), PAGE_SIZE);
> + if (*end) {
> +  fprintf(stderr, usage);
> +  exit(1);
> + }
> + data_size = ALIGN(memparse(argv[3], &end), PAGE_SIZE);
> + if (*end) {
> +  fprintf(stderr, usage);
> +  exit(1);
> + }
> +
> + /* retrieve group name */
> + mygroup = getenv("MYGROUP");
> + if (!mygroup) {
> +  fprintf(stderr,
> +   "ERROR: undefined environment variable MYGROUP\n");
> +  exit(1);
> + }
> +
> + children = malloc(sizeof(pid_t) * threads);
> + if (!children) {
> +  fprintf(stderr, "ERROR: not enough memory\n");
> +  exit(1);
> + }
> +
> + /* handle user interrupt */
> + signal(SIGINT, signal_handler);
> + /* handle kill from shell */
> + signal(SIGTERM, signal_handler);
> +
> + fprintf(stdout, "chunk_size %luKiB, data_size %luKiB\n",
> +  KB(chunk_size), KB(data_size));
> + fflush(stdout);
> +
> +       gettimeofday(&start, NULL);
```

> +	for (i = 0; i < threads ; i++)
> +		spawn(i);
> +	for (i = 0; i < threads; i++) {
> +		int status;
> +		wait(&status);
> +		if (!WIFEXITED(status))
> +			exit(1);
> +	}
> +	gettimeofday(&stop, NULL);
> +
> +	timersub(&stop, &start, &diff);
> + print_results(0, NUM_IOPS, data_size * threads * NUM_IOPS, &diff);
> + fflush(stdout);
> + free(children);
> +
> +	exit(0);
> +}
> diff --exclude CVS -urpN ltp/testcases/kernel/controllers.orig/io-throttle/io_throttle_testplan.txt
ltp/testcases/kernel/controllers/io-throttle/io_throttle_testplan.txt
> --- ltp/testcases/kernel/controllers.orig/io-throttle/io_throttle_testplan.txt 1970-01-01
01:00:00.000000000 +0100
> +++ ltp/testcases/kernel/controllers/io-throttle/io_throttle_testplan.txt 2008-07-07
11:25:25.000000000 +0200
> @@ -0,0 +1,36 @@
> +The I/O bandwidth controller testplan includes a complete set of testcases to
> +verify the effectiveness of the block device I/O throttling capabilities for
> +cgroups.
> +
> +I/O bandwidth limitations are imposed by the testcase script and verified doing
> +I/O activity on a limited block device. Tests are supposed to be passed if the
> +I/O rates of all the different workloads always respect the I/O limitations.
> +
> +TESTCASE DESCRIPTION:
> +====================
> +First of all we evaluate the physical I/O bandwidth (physical-io-bw) of the
> +block device where the current working directory resides.
> +
> +Based on the physical I/O bandwidth three cgroups are created: cgroup-1,
> +cgroup-2, cgroup-3. Cgroups use respectively the following I/O bandwidth
> +limitations:
> +- cgroup-1: physical-io-bw / 2
> +- cgroup-2: physical-io-bw / 4
> +- cgroup-3: physical-io-bw / 8
> +
> +Each test is considered passed only if the I/O limitations above are respected.
> +
> +Currently the following different scenarios are tested:
> +- 1 single stream per cgroup using leaky-bucket I/O throttling

> +- 1 single stream per cgroup using token-bucket I/O throttling
> +- 2 parallel streams per cgroup using leaky-bucket I/O throttling
> +- 2 parallel streams per cgroup using token-bucket I/O throttling
> +- 4 parallel streams per cgroup using leaky-bucket I/O throttling
> +- 4 parallel streams per cgroup using token-bucket I/O throttling
> +
> +For any other information please refer to
> +Documentation/controllers/io-throttle.txt in kernel documentation.
> +
> +Questions?
> +----------
> +Send email to: righi.andrea@gmail.com
> diff --exclude CVS -urpN ltp/testcases/kernel/controllers.orig/io-throttle/Makefile
ltp/testcases/kernel/controllers/io-throttle/Makefile
> --- ltp/testcases/kernel/controllers.orig/io-throttle/Makefile 1970-01-01 01:00:00.000000000
+0100
> +++ ltp/testcases/kernel/controllers/io-throttle/Makefile 2008-07-03 19:24:17.000000000 +0200
> @@ -0,0 +1,16 @@
> +CFLAGS += -Wall
> +CPPFLAGS += -I../../../../include -I../libcontrollers
> +LDLIBS += -lm -L../../../../lib/ -L../libcontrollers -lcontrollers -lltp
> +
> +SRCS   = $(wildcard *.c)
> +
> +TARGETS = $(patsubst %.c,%,$(SRCS))
> +
> +all: $(TARGETS)
> +
> +clean:
> + rm -f $(TARGETS) *.o
> +
> +install:
> + @set -e; for i in $(TARGETS) run_io_throttle_test.sh myfunctions.sh; do ln -f $$i ../../../bin/$$i ;
chmod +x $$i ; done
> +
> diff --exclude CVS -urpN ltp/testcases/kernel/controllers.orig/io-throttle/myfunctions.sh
ltp/testcases/kernel/controllers/io-throttle/myfunctions.sh
> --- ltp/testcases/kernel/controllers.orig/io-throttle/myfunctions.sh 1970-01-01
01:00:00.000000000 +0100
> +++ ltp/testcases/kernel/controllers/io-throttle/myfunctions.sh 2008-07-07 10:41:01.000000000
+0200
> @@ -0,0 +1,61 @@
> +#!/bin/sh
> +#
> +# This program is free software; you can redistribute it and/or
> +# modify it under the terms of the GNU General Public
> +# License as published by the Free Software Foundation; either
> +# version 2 of the License, or (at your option) any later version.

```
> +#
> +# This program is distributed in the hope that it will be useful,
> +# but WITHOUT ANY WARRANTY; without even the implied warranty of
> +# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
> +# General Public License for more details.
> +#
> +# You should have received a copy of the GNU General Public
> +# License along with this program; if not, write to the
> +# Free Software Foundation, Inc., 59 Temple Place - Suite 330,
> +# Boston, MA 021110-1307, USA.
> +#
> +# Copyright (C) 2008 Andrea Righi <righi.andrea@gmail.com>
> +#
> +# usage . myfunctions.sh
> +
> +setup()
> +{
> + # create testcase cgroups
> + if [ -e /dev/blockioctl ]; then
> +  echo "WARN: /dev/blockioctl already exist! overwriting."
> +  cleanup
> + fi
> + mkdir /dev/blockioctl
> + mount -t cgroup -o blockio cgroup /dev/blockioctl
> + if [ $? -ne 0 ]; then
> +  echo "ERROR: could not mount cgroup filesystem " \
> +   " on /dev/blockioctl. Exiting test."
> +  cleanup
> +  exit 1
> + fi
> + for i in `seq 1 3`; do
> +  if [ -e /dev/blockioctl/cgroup-$i ]; then
> +   rmdir /dev/blockioctl/cgroup-$i
> +   echo "WARN: earlier cgroup-$i found and removed"
> +  fi
> +  mkdir /dev/blockioctl/cgroup-$i
> +  if [ $? -ne 0 ]; then
> +   echo "ERROR: could not create cgroup-$i" \
> +    "Check your permissions. Exiting test."
> +   cleanup
> +   exit 1
> +  fi
> + done
> +}
> +
> +cleanup()
> +{
> + echo "Cleanup called"
```

> + for i in `seq 1 3`; do
> + rmdir /dev/blockioctl/cgroup-$i
> + rm -f /tmp/cgroup-$i.out
> + done
> + umount /dev/blockioctl
> + rmdir /dev/blockioctl
> +}
> diff --exclude CVS -urpN ltp/testcases/kernel/controllers.orig/io-throttle/README
ltp/testcases/kernel/controllers/io-throttle/README
> --- ltp/testcases/kernel/controllers.orig/io-throttle/README 1970-01-01 01:00:00.000000000
+0100
> +++ ltp/testcases/kernel/controllers/io-throttle/README 2008-07-07 11:14:57.000000000 +0200
> @@ -0,0 +1,56 @@
> +TEST SUITE:
> +
> +The directory io-throttle contains the tests related to block device I/O
> +bandwdith controller.
> +
> +More testcases are expected to be added in future.
> +
> +TESTS AIM:
> +
> +The aim of the tests is to check the block device I/O throttling functionality
> +for cgroups.
> +
> +FILES DESCRIPTION:
> +
> +iobw.c
> +---------------
> +Simple benchmark to generate parallel streams of direct I/O (O_DIRECT). This
> +benchmark fork()s one task per stream. Each task creates a separate file in the
> +current working directory, fills it with data using O_DIRECT writes and re-read
> +the whole file always in O_DIRECT mode. Different timestamps are used to
> +evaluate per-task I/O rate and total I/O rate (seen by the parent).
> +
> +myfunctions.sh
> +----------
> +This file contains the functions which are common for the io-throttle tests.
> +For ex. the setup and cleanup functions which do the setup for running the
> +test and do the cleanup once the test finishes. The setup() function creates
> +/dev/blockioctl directory and mounts cgroup filesystem on it with memory
> +controller. It then creates a number(n) of groups in /dev/blockioctl. The
> +cleanup function does a complete cleanup of the system.
> +
> +Most of the error scenarios have been taken care of for a sane cleanup of the
> +system. However if cleanup fails in any case, just manually execute the
> +commands written in cleanup function in myfunctions.sh.
> +One of the most common causes of failed cleanup is that you have done cd into

> +any of the groups in controller dir tree.
> +
> +run_io_throttle_test.sh
> +------------------
> +This script creates different scenarios for I/O bandwidth controller testing
> +and fires (n) tasks in different groups to write and read different I/O streams
> +etc. It waits for the return status from tasks and reports test pass/fail
> +accordingly.
> +
> +Makefile
> +--------
> +The usual makefile for this directory
> +
> +PASS/FAIL CRITERION:
> +====================
> +The test cases are intelligent enough in deciding the pass or failure of a
> +test.
> +
> +README:
> +--------
> +This file.
> diff --exclude CVS -urpN ltp/testcases/kernel/controllers.orig/io-throttle/run_io_throttle_test.sh
ltp/testcases/kernel/controllers/io-throttle/run_io_throttle_test.sh
> --- ltp/testcases/kernel/controllers.orig/io-throttle/run_io_throttle_test.sh 1970-01-01
01:00:00.000000000 +0100
> +++ ltp/testcases/kernel/controllers/io-throttle/run_io_throttle_test.sh 2008-07-07
11:33:04.000000000 +0200
> @@ -0,0 +1,114 @@
> +#!/bin/bash
> +#
> +# This program is free software; you can redistribute it and/or
> +# modify it under the terms of the GNU General Public
> +# License as published by the Free Software Foundation; either
> +# version 2 of the License, or (at your option) any later version.
> +#
> +# This program is distributed in the hope that it will be useful,
> +# but WITHOUT ANY WARRANTY; without even the implied warranty of
> +# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
> +# General Public License for more details.
> +#
> +# You should have received a copy of the GNU General Public
> +# License along with this program; if not, write to the
> +# Free Software Foundation, Inc., 59 Temple Place - Suite 330,
> +# Boston, MA 021110-1307, USA.
> +#
> +# Copyright (C) 2008 Andrea Righi <righi.andrea@gmail.com>
> +#
> +# Usage: ./run_io_throttle_test.sh

```
> +# Description: test block device I/O bandwidth controller functionalities
> +
> +. myfunctions.sh
> +
> +trap cleanup SIGINT
> +
> +BUFSIZE=16m
> +DATASIZE=64m
> +
> +setup
> +
> +# get the device name of the entire mounted block device
> +dev=`df -P . | sed '1d' | cut -d' ' -f1 | sed 's/[p]*[0-9]*$//'`
> +
> +# evaluate device bandwidth
> +export MYGROUP=
> +phys_bw=`./iobw -direct 1 $BUFSIZE $DATASIZE | grep TOTAL | awk '{print $7}'`
> +if [ $? -ne 0 ]; then
> + echo "ERROR: could not evaluate i/o bandwidth of $dev. Exiting test."
> + cleanup
> + exit 1
> +fi
> +echo ">> physical i/o bandwidth limit is: $phys_bw KiB/s"
> +# show cgroup i/o bandwidth limits
> +for i in `seq 1 3`; do
> + MYGROUP=cgroup-$i
> + echo "($MYGROUP) max i/o bw: " \
> +  "$(($phys_bw / `echo 2^$i | bc`)) KiB/s + O_DIRECT"
> +done
> +
> +for tasks in 1 2 4; do
> +for strategy in 0 1; do
> + # set bw limiting rules
> + for i in `seq 1 3`; do
> +  limit=$(($phys_bw * 1024 / `echo 2^$i | bc`))
> +  IOBW[$i]=$(($limit / 1024))
> +  /bin/echo $dev:$limit:$strategy:$limit > \
> +   /dev/blockioctl/cgroup-$i/blockio.bandwidth
> +  if [ $? -ne 0 ]; then
> +   echo "ERROR: could not set i/o bandwidth limit for cgroup-$i. Exiting test."
> +   cleanup
> +   exit 1
> +  fi
> + done
> +
> + # run benchmark
> + if [ $tasks -eq 1 ]; then
> +  stream="stream"
```

```
> + else
> +  stream="streams"
> + fi
> + echo -n ">> testing $tasks parallel $stream per cgroup "
> + if [ $strategy -eq 0 ]; then
> +  echo "(leaky-bucket i/o throttling)"
> + else
> +  echo "(token-bucket i/o throttling)"
> + fi
> + for i in `seq 1 3`; do
> +  MYGROUP=cgroup-$i
> +  /bin/echo $$ > /dev/blockioctl/$MYGROUP/tasks
> +  if [ $? -ne 0 ]; then
> +   echo "ERROR: could not set i/o bandwidth limit for cgroup-$i. Exiting test."
> +   cleanup
> +   exit 1
> +  fi
> +  # exec i/o benchmark
> +  ./iobw -direct $tasks $BUFSIZE $DATASIZE > /tmp/$MYGROUP.out &
> +  PID[$i]=$!
> + done
> + /bin/echo $$ > /dev/blockioctl/tasks
> +
> + # wait for children completion
> + for i in `seq 1 3`; do
> +  MYGROUP=cgroup-$i
> +  wait ${PID[$i]}
> +  ret=$?
> +  if [ $ret -ne 0 ]; then
> +   echo "ERROR: error code $ret during test $tasks.$strategy.$i. Exiting test."
> +   cleanup
> +   exit 1
> +  fi
> +  iorate=`grep parent /tmp/${MYGROUP}.out | awk '{print $7}'`
> +  diff=$((${IOBW[$i]} - $iorate))
> +  echo "($MYGROUP) i/o-bw ${IOBW[$i]} KiB/s, i/o-rate $iorate KiB/s, err $diff KiB/s"
> +  if [ ${IOBW[$i]} -ge $iorate ]; then
> +   echo "TPASS   Block device I/O bandwidth controller: test $tasks.$strategy.$i PASSED";
> +  else
> +   echo "TFAIL   Block device I/O bandwidth controller: test $tasks.$strategy.$i FAILED";
> +  fi
> + done
> +done
> +done
> +
> +cleanup
> diff --exclude CVS -urpN ltp/testcases/kernel/controllers.orig/Makefile
ltp/testcases/kernel/controllers/Makefile
```

> --- ltp/testcases/kernel/controllers.orig/Makefile 2008-04-30 09:19:29.000000000 +0200
> +++ ltp/testcases/kernel/controllers/Makefile 2008-07-07 11:43:07.000000000 +0200
> @@ -1,6 +1,7 @@
> SUBDIRS = libcontrollers cpuctl memctl
> CHECK_CPUCTL = $(shell grep -w cpu /proc/cgroups|cut -f1)
> CHECK_MEMCTL = $(shell grep -w memory /proc/cgroups|cut -f1)
> +CHECK_BLOCKIOCTL= $(shell grep -w blockio /proc/cgroups|cut -f1)
> all:
>   @set -e;
> ifeq ($(CHECK_CPUCTL),cpu)
> @@ -17,6 +18,13 @@ else
>   echo "Kernel is not compiled with memory resource controller support";
> endif
>
> +ifeq ($(CHECK_BLOCKIOCTL),blockio)
> +
> + for i in $(SUBDIRS); do $(MAKE) -C $$i $@ ;done;
> +else
> + echo "Kernel is not compiled with blockio resource controller support";
> +endif
> +
> install:
>   @set -e; \
> ln -f test_controllers.sh ../../bin/test_controllers.sh;
> @@ -37,5 +45,13 @@ else
>   echo "Kernel is not compiled with memory resource controller support";
> endif
>
> +ifeq ($(CHECK_BLOCKIOCTL),blockio)
> +
> + for i in $(SUBDIRS); do $(MAKE) -C $$i install ; done; \
> + chmod ugo+x test_controllers.sh;
> +else
> + echo "Kernel is not compiled with blockio resource controller support";
> +endif
> +
> clean:
>   @set -e; for i in $(SUBDIRS); do $(MAKE) -C $$i clean ; done
> diff --exclude CVS -urpN ltp/testcases/kernel/controllers.orig/test_controllers.sh
ltp/testcases/kernel/controllers/test_controllers.sh
> --- ltp/testcases/kernel/controllers.orig/test_controllers.sh 2008-05-26 13:26:44.000000000
+0200
> +++ ltp/testcases/kernel/controllers/test_controllers.sh 2008-07-07 11:41:09.000000000 +0200
> @@ -38,6 +38,7 @@ if [-f /proc/cgroups]
> then
>   CPU_CONTROLLER=`grep -w cpu /proc/cgroups | cut -f1`;
>   MEM_CONTROLLER=`grep -w memory /proc/cgroups | cut -f1`;
> + IOTHROTTLE_CONTROLLER=`grep -w blockio /proc/cgroups | cut -f1`;

>
>   if ["$CPU_CONTROLLER" = "cpu"]
>   then
> @@ -68,6 +69,15 @@ then
>   echo "Kernel does not support for memory controller";
>   echo "Skipping all memory controller testcases....";
>   fi
> +
> + if ["$IOTHROTTLE_CONTROLLER" = "blockio"]
> + then
> + $LTPROOT/testcases/bin/run_memctl_test.sh 1;
> + else
> + echo "CONTROLLERS TESTCASES: WARNING";
> + echo "Kernel does not support blockio controller";
> + echo "Skipping all block device I/O throttling testcases....";
> + fi
> else
>   echo "CONTROLLERS TESTCASES: WARNING"
>   echo "Kernel does not support for control groups";

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers