用例标题

CPU指令架构支持-CPU内置功能-d.物理多路 CPU-3)支持NUMA架构，提供NUMA驱动编程接口，根据ACPI描述的NUMA架构内存信息进行进程调度优化，让进程运行在本NUMA节点之上

用例描述

使用membind策略在进程的内存绑定节点上面分配本地内存遍历系统所有内存节点，对每个节点执行如下操作：

1、用numactl将进程support\_numa\_case1绑定到当前节点上同时内存分配按顺序循环绑定到除当前节点之外的节点上，进程support\_numa\_case1会在内存绑定节点上面分配1MB的内存

2、睡眠两秒待进程运行稳定

3、将进程与当前节点解绑定，使用taskset让进程运行在系统所有核上

4、睡眠10秒等待进程自动迁移到1MB内存所在的节点上

5、获取进程当前所在的核编号判断核编号是否在1MB内存所在节点的核列表中

6、如果在则NUMA自动迁移进程到本地节点成功

7、对于当前节点，只要除当前节点之外的任意内存绑定节点满足上述操作则成功

8、对于当前节点，如果除当前节点之外的所有内存绑定节点都不满足上述操作则用例失败。

9、遍历系统所有内存节点之后如果所有的节点都操作成功则用例成功。

用例步骤

1、yum install numactl-devel numactl

安装成功

2、gcc support\_numa\_case1.c -lpthread -o support\_numa\_case1

成功生成support\_numa\_case1文件

3、gcc support\_numa.c -lpthread -o support\_numa

成功生成support\_numa文件

4、执行./numa.sh

打印"PASS NUMA local node and memory affinity"则用例执行成功

相关测试代码：

用例包含三个源码文件: numa.sh、support\_numa.c、 tst\_getconf.c

* numa.sh 包含了测试用例
* support\_numa.c 分配1M物理内存
* support\_numa\_case1.c 在指定节点分配1M物理内存，然后将别的节点的主线程自动迁移到指定节点
* tst\_getconf.c  读取系统配置参数PAGESIZE
* Makefile   support\_numa，support\_numa\_case1和tst\_getconf命令的编译脚本

使用方法

编译：make

执行：./numa.sh

PS：如果numa.sh没有可执行权限，则添加可执行权限chmod a+x numa.sh

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* \*/

/\* File: support\_numa.c \*/

/\* \*/

/\* \*/

/\* Author: guohui \*/

/\* \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdio.h>

#include <stdlib.h>

#include <errno.h>

#include <unistd.h>

#include <signal.h>

#include <limits.h>

#include <string.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <sys/mman.h>

#include <fcntl.h>

#define MB (1<<20)

#define PAGE\_SIZE getpagesize()

#define barrier() \_\_asm\_\_ \_\_volatile\_\_("": : :"memory")

static void help(void)

{

printf("Input: Describe input arguments to this program\n");

printf(" argv[1] == \"alloc\_1MB\" then allocate 1MB of memory\n");

printf("Exit: On failure - Exits with non-zero value\n");

printf(" On success - exits with 0 exit value\n");

exit(1);

}

int main(int argc, char \*argv[])

{

int i, hpsz;

char \*buf = NULL;

if (argc != 2) {

fprintf(stderr, "Here expect only one number(i.e. 2) as the parameter\n");

exit(1);

}

if (!strcmp(argv[1], "alloc\_1MB")) {

buf = malloc(MB);

if (!buf) {

fprintf(stderr, "Memory is not available\n");

exit(1);

}

for (i = 0; i < MB; i += PAGE\_SIZE) {

buf[i] = 'a';

barrier();

}

raise(SIGSTOP);

free(buf);

} else {

help();

}

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* \*/

/\* File: support\_numa\_case1.c \*/

/\* \*/

/\* Description: 分配1MB大小的内存 \*/

/\* \*/

/\* Author: guohui \*/

/\* \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define \_GNU\_SOURCE

#include <stdio.h>

#include <stdlib.h>

#include <errno.h>

#include <unistd.h>

#include <signal.h>

#include <limits.h>

#include <string.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <sys/mman.h>

#include <fcntl.h>

#include <pthread.h>

#include <sys/syscall.h>

#include <numaif.h>

#define MB (1<<20)

#define PAGE\_SIZE getpagesize()

#define barrier() \_\_asm\_\_ \_\_volatile\_\_("": : :"memory")

#define gettid() syscall(\_\_NR\_gettid)

static char \*buf = NULL;

static int p1[2];

static int p2[2];

static char read\_buff1[8];

static char read\_buff2[8];

static int should\_run = 1;

static void help(void)

{

printf("Input: Describe input arguments to this program\n");

printf(" argv[1] == \"alloc\_1MB\" then allocate 1MB of memory\n");

printf(" argv[2] == \"cpus\_list\" then bind sub thread to those cpus\n");

printf("Exit: On failure - Exits with non-zero value\n");

printf(" On success - exits with 0 exit value\n");

exit(1);

}

static void delay(void)

{

int i = 4600;

while (i-- > 0);

}

static void sig\_usr(int signo)

{

if (signo == SIGUSR1)

{

should\_run = 0;

exit(0);

}

}

static int is\_digitstr(char \*str)

{

return (strspn(str, "0123456789")==strlen(str));

}

static void \* pthread(void \*arg)

{

char \*cpus = (char \*)arg;

int cpu;

cpu\_set\_t mask;

int tid = gettid();

int i;

char v = 'a';

char \*p;

char tmp[256] = { 0 };

strncpy(tmp, cpus, strlen(cpus));

CPU\_ZERO(&mask);

p = strtok(tmp, " ");

while (p) {

if (!is\_digitstr(p)) {

p = strtok(NULL, " ");

continue;

}

cpu = atoi(p);

CPU\_SET(cpu, &mask);

p = strtok(NULL, " ");

}

if(sched\_setaffinity(tid, sizeof(mask), &mask) < 0) {

perror("pthread\_setaffinity\_np failed.\n");

}

while (should\_run) {

read(p1[1], read\_buff2, 1);

sleep(1);

for (i = 0; i < MB; i += PAGE\_SIZE) {

buf[i] = 'a';

barrier();

}

v += 1;

if (v > 'z')

v = 'a';

write(p2[1], "1", 1);

}

return NULL;

}

static void \* pthread\_node(void \*arg)

{

char \*cpus = (char \*)arg;

int cpu;

cpu\_set\_t mask;

int tid = gettid();

int i;

char v = 'a';

char \*p;

unsigned long count = 0;

char tmp[256] = { 0 };

strncpy(tmp, cpus, strlen(cpus));

CPU\_ZERO(&mask);

p = strstr(tmp, " ");

if (p)

\*p = '\0';

cpu = atoi(tmp);

CPU\_SET(cpu, &mask);

if(sched\_setaffinity(tid, sizeof(mask), &mask) < 0) {

perror("pthread\_setaffinity\_np failed.\n");

}

while (should\_run) {

read(p1[1], read\_buff2, 1);

sleep(10);

for (i = 0; i < MB; i += PAGE\_SIZE) {

buf[i] = 'a';

barrier();

}

v += 1;

if (v > 'z')

v = 'a';

write(p2[1], "1", 1);

}

return NULL;

}

static void \* pthread\_local(void \*arg)

{

char \*cpus = (char \*)arg;

int cpu;

cpu\_set\_t mask;

int tid = gettid();

int i;

char v = 'a';

char \*p;

char tmp[256] = { 0 };

strncpy(tmp, cpus, strlen(cpus));

CPU\_ZERO(&mask);

p = strtok(tmp, " ");

if (!is\_digitstr(p)) {

p = strtok(NULL, " ");

}

cpu = atoi(p);

CPU\_SET(cpu, &mask);

if (sched\_setaffinity(tid, sizeof(mask), &mask) < 0) {

perror("pthread\_setaffinity\_np failed.\n");

}

while (should\_run) {

read(p1[1], read\_buff2, 1);

sleep(1);

while (1) {

barrier();

}

write(p2[1], "1", 1);

}

return NULL;

}

int main(int argc, char \*argv[])

{

int i, hpsz;

pthread\_t tidp;

char v='a';

char \*tmp;

cpu\_set\_t mask;

if (!strcmp(argv[1], "alloc\_1MB")) {

pipe(p1);

pipe(p2);

buf = malloc(MB);

if (!buf) {

fprintf(stderr, "Memory is not available\n");

exit(1);

}

if (signal(SIGUSR1, sig\_usr) == SIG\_ERR) {

printf("sig\_usr signal error!\n");

}

for (i = 0; i < MB; i += PAGE\_SIZE) {

buf[i] = 'a';

barrier();

}

tmp = argv[2];

while (tmp) {

tmp += 1;

if (pthread\_create(&tidp, NULL, pthread, (void \*)argv[2]) == -1) {

printf("create error!\n");

return 1;

}

tmp = strstr(tmp, " ");

}

tmp = argv[3];

while (tmp) {

if (pthread\_create(&tidp, NULL, pthread\_local, (void \*)tmp) == -1) {

printf("create error!\n");

return 1;

}

tmp = strstr(tmp, " ");

if (!tmp)

break;

tmp += 1;

}

while (should\_run) {

delay();

for (i = 0; i < MB; i += PAGE\_SIZE) {

buf[i] = 'a';

barrier();

}

v += 1;

if (v > 'z')

v = 'a';

write(p1[0], "1", 1);

read(p2[0], read\_buff1, 1);

}

free(buf);

} else {

help();

}

return 0;

}

/\* tst\_getconf

\* Copyright (c) 2018 Myl猫ne Josserand <mylene.josserand@bootlin.com>

\*

\*/

#include <stdio.h>

#include <unistd.h>

#include <string.h>

static void print\_help(void)

{

printf("Usage: tst\_getconf variable\n\n");

printf(" variable: can be PAGESIZE/PAGE\_SIZE");

printf(" or \_NPROCESSORS\_ONLN (for the moment)\n\n");

printf("example: tst\_getconf PAGESIZE\n");

}

int main(int argc, char \*argv[])

{

int opt;

while ((opt = getopt(argc, argv, ":h")) != -1) {

switch (opt) {

case 'h':

print\_help();

return 0;

default:

print\_help();

return 1;

}

}

if (argc != 2) {

print\_help();

return 1;

}

if (!strcmp(argv[optind], "\_NPROCESSORS\_ONLN")) {

printf("%ld\n", sysconf(\_SC\_NPROCESSORS\_ONLN));

} else if (!strcmp(argv[optind], "PAGESIZE") ||

!strcmp(argv[optind], "PAGE\_SIZE")) {

printf("%ld\n", sysconf(\_SC\_PAGE\_SIZE));

} else {

printf("tst\_getconf: Unrecognized variable \'%s\'\n",

argv[optind]);

return -1;

}

return 0;

}

# numa.sh

#!/bin/sh

#

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*#

#\* \*#

#\* File: Makefile \*#

#\* \*#

#\* Description: 测试用例执行脚本 \*#

#\* Test #1: 验证 cpunodebind 和 membind 绑定测略 \*#

#\* 验证进程运行在本NUMA节点并在本NUMA节点分配内存 \*#

#\* Test #2: 验证 preferred node bind 绑定内存测略 \*#

#\* 验证跨路内存访问，其他路CPU访问本地内存 \*#

#\* Test #3: 验证 memory interleave on all nodes 测略 \*#

#\* Test #4: 验证 localalloc 分配测略 \*#

#\* \*#

#\* Author: guohui \*#

#\* \*#

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*#

# $1 - Pid number

# $2 - Node number

get\_node\_index()

{

local pid=$1

local nid="Node $2"

echo $(numastat -p $pid | sed '3q;d' | awk -F '[[:space:]][[:space:]]+' \

-v node="$nid" '{ for (i = 1; i <= NF; ++i) if($i==node) print i; exit }')

}

# 转换M为B字节.

# $1 - Pid number

# $2 - Node number

# $3 - Size for multiplication (e.g. 1024, $MB)

get\_mem\_cur()

{

local pid=$1

local index=$(echo "$(get\_node\_index $pid $2)")

local size=$3

local numstat=$(numastat -p $pid |awk '/^Total/ {print $'$index'}')

if [ -z "$numstat" ]; then

echo 0

return

fi

echo $(echo "$numstat \* $size" | bc)

}

get\_node\_cpus()

{

local ne=$1

ne=$(echo "$ne+1" | bc)

echo $(numactl --hardware | grep "cpus:" | awk -F ':' -v node="$ne" '{ if (NR == node){print $2;} }')

}

setup()

{

export MB=$((1024\*1024))

export PAGE\_SIZE=$(./tst\_getconf PAGESIZE)

export CPU\_MAX\_INDEX=$(echo "$(./tst\_getconf \_NPROCESSORS\_ONLN)-1" | bc)

total\_nodes=0

nodes\_list=$(numactl --show | grep nodebind | cut -d ':' -f 2)

node\_max=0

for node in $nodes\_list; do

total\_nodes=$((total\_nodes+1))

node\_max=$(echo $(($node\_max>$node?$node\_max:$node)))

done

echo "The system contains $total\_nodes nodes: $nodes\_list"

if [ $total\_nodes -le 1 ]; then

echo "SUT does not support NUMA policy or not a NUMA machine"

exit

fi

}

test1()

{

local ret

local i=0

echo "verifing please wait ..."

for node in $nodes\_list; do

local cnt=1

local flags=0

echo "current node = " $node

for ne in $nodes\_list; do

if [ $ne -eq $node ]; then

continue;

fi

local cpus\_list=$(get\_node\_cpus $ne)

local cpus\_local=$(get\_node\_cpus $node)

numactl --cpunodebind=$node --membind=$ne ./support\_numa\_case1 alloc\_1MB "$cpus\_list" "$cpus\_local" &

pid=$!

sleep 2

taskset -pc 0-$CPU\_MAX\_INDEX $pid >/dev/null 2>&1

sleep 15

running\_on\_cpu=$(awk '{ print $39; }' /proc/$pid/stat)

for cpu in $cpus\_list; do

if [ $(echo "$running\_on\_cpu == $cpu" | bc) -eq 1 ]; then

flags=1

break;

fi

done

if [ $flags -eq 1 ]; then

kill -USR1 $pid >/dev/null 2>&1

break;

fi

if [ $flags -eq 0 ]&&[ $ne -eq $node\_max ]; then

echo "Process running on cpu $running\_on\_cpu but expected to run on cpu $cpus\_list"

kill -USR1 $pid >/dev/null 2>&1

return

fi

kill -USR1 $pid >/dev/null 2>&1

done

done

echo "PASS NUMA local node and memory affinity"

}

test2()

{

local mem\_curr

for node in $nodes\_list; do

local cnt=1

for ne in $nodes\_list; do

numactl --cpunodebind=$node --preferred=$ne ./support\_numa alloc\_1MB &

pid=$!

mem\_curr=$(get\_mem\_cur $pid $ne $MB)

if [ $(echo "$mem\_curr < $MB" |bc ) -eq 1 ]; then

echo "NUMA preferred memory allocated in node $Preferred\_node is less than expected"

kill -CONT $pid >/dev/null 2>&1

return

fi

cnt=$((cnt+1))

kill -CONT $pid >/dev/null 2>&1

done

done

echo "PASS NUMA preferred node policy"

}

test3()

{

local mem\_curr

Exp\_incr=$(echo "$MB / $total\_nodes" |bc)

numactl --interleave=all ./support\_numa alloc\_1MB &

pid=$!

for node in $nodes\_list; do

mem\_curr=$(get\_mem\_cur $pid $node $MB)

if [ $(echo "$mem\_curr < $Exp\_incr" |bc ) -eq 1 ]; then

echo "NUMA interleave memory allocated in node$node is less than expected"

kill -CONT $pid >/dev/null 2>&1

return

fi

done

kill -CONT $pid >/dev/null 2>&1

echo "PASS NUMA interleave policy"

}

test4()

{

local mem\_curr

for node in $nodes\_list; do

numactl --cpunodebind=$node --localalloc ./support\_numa alloc\_1MB &

pid=$!

mem\_curr=$(get\_mem\_cur $pid $node $MB)

if [ $(echo "$mem\_curr < $MB" |bc ) -eq 1 ]; then

echo "NUMA localnode memory allocated in node $node is less than expected"

kill -CONT $pid >/dev/null 2>&1

return

fi

kill -CONT $pid >/dev/null 2>&1

done

echo "PASS NUMA local node allocation"

}

setup

test1

test2

# test3

# test4

set +m killall support\_numa >/dev/null 2>&1

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*#

#\* \*#

#\* File: Makefile \*#

#\* \*#

#\* Description: 测试用例编译脚本 \*#

#\* \*#

#\* Author: guohui \*#

#\* \*#

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*#

CC := gcc

LDFLAGS := -lnuma -lpthread

SRCS = $(wildcard \*.c)

OBJS1 := support\_numa.o

OBJS2 := support\_numa\_case1.o

OBJS3 := tst\_getconf.o

TARGET1 := support\_numa

TARGET2 := support\_numa\_case1

TARGET3 := tst\_getconf

.PHONY:all

all:$(TARGET1) $(TARGET2) $(TARGET3)

%.o : %.c

$(CC) -c -o $@ $<

$(TARGET1): $(OBJS1)

$(CC) $(LDFLAGS) -o $(TARGET1) $(OBJS1)

$(TARGET2): $(OBJS2)

$(CC) $(LDFLAGS) -o $(TARGET2) $(OBJS2)

$(TARGET3): $(OBJS3)

$(CC) -o $(TARGET3) $(OBJS3)

clean:

rm -f $(shell find -name "\*.o")

rm -f $(TARGET1)

rm -f $(TARGET2)

rm -f $(TARGET3)

©统信软件技术有限公司。访问者可将本网站提供的内容或服务用于个人学习、研究或欣赏，以及其他非商业性或非盈利性用途，但同时应遵守著作权法及其他相关法律的规定，不得侵犯本网站及相关权利人的合法权利。除此以外，将本网站任何内容或服务进行转载，须备注：**该文档出自【faq.uniontech.com】统信软件知识分享平台**。否则统信软件将追究相关版权责任。